EXHIBIT D

Redbooks
ibm.com/redbooks

Creating IBM z/OS Cloud Services

Jeffrey Bisti

Frank De Gilio

Randy Frerking

Rich Jackson

Charlie Lawrence

Kellie Mathis

Cloud

z Systems

IBM

Redbooks

International Technical Support Organization

Creating IBM z/OS Cloud Services

February 2016

SG24-8324-00

Note: Before using this information and the product it supports, read the information in “Notices” on page v.

First Edition (February 2016)

This edition applies to Version 2, Release 1, Modification 0 of IBM z/OS (product number 5650-ZOS).

Contents

Notices

Trademarks

Find and read thousands of IBM Redbooks publications

- Search, bookmark, save and organize favorites
- Get personalized notifications of new content
- Link to the latest Redbooks blogs and videos

Get the latest version of the Redbooks Mobile App

iOS **Download Now** Android

Promote your business in an IBM Redbooks publication

Place a Sponsorship Promotion in an IBM® Redbooks® publication, featuring your business or solution with a link to your web site.

Qualified IBM Business Partners may place a full page promotion in the most popular Redbooks publications. Imagine the power of being seen by users who download millions of Redbooks publications each year!

ibm.com/Redbooks
About Redbooks → Business Partner Programs

THIS PAGE INTENTIONALLY LEFT BLANK

Preface

This IBM® Redbooks® publication discusses the real world experience of an enterprise that developed and implemented IBM z/OS® cloud services. This book shares the experience of a team at Walmart Technology, Walmart Stores, Inc.® and some of the decisions they made to create business critical cloud services. These experiences and approaches relate to the z/OS platform, and might not apply to other hybrid cloud approaches.

This book highlights the strengths and characteristics of z/OS that led the Walmart infrastructure and software engineers to use this platform as they transitioned from a traditional IT deployment to a cloud model.

Embarking on a cloud strategy can be overwhelming. No shortage of approaches to cloud computing exists. This book focuses on a pragmatic approach for enterprises that are struggling to take advantage of their business assets in the cloud.

This book introduces the basic cloud concepts as defined by the National Institute of Standards and Technology (NIST). Each chapter explains the importance of a particular NIST characteristic, the z/OS role in accomplishing the characteristic, and how it was implemented by the Walmart Technology team.

This book covers the operational aspects that should be addressed to provide z/OS cloud services:

- Chapter 1. Introduction
- Chapter 2. Cloud from a services perspective
- Chapter 3. Shared pool: Configurable resources
- Chapter 4. Elastic environment
- Chapter 5. Ubiquitous access: ReST enabling your runtime
- Chapter 6. Metering and measuring
- Chapter 7. Self-service provisioning

This book is intended for IT professionals who are considering extending their IBM z Systems™ environment to a hybrid cloud by unleashing the power of cloud services on z/OS.

For information about creating cloud services that are hosted in IBM CICS®, see *How Walmart Became a Cloud Services Provider with IBM CICS*, SG24-8347.

Authors

This book was produced by a team of specialists from IBM and Walmart Technology, Walmart Stores Inc.

Figure 1 From left to right Rich Jackson, Randy Frerking, Charlie Lawrence, Kellie Mathis, Frank De Gilio, Jeffrey Bisti

Jeffrey Bisti is a Cloud Solutions Architect working at the IBM World Wide Client Technology Center. He has applied his Linux and z/OS technical background to several projects dealing with building and maintaining large-scale storage and IP networks. In his current assignment, Jeffrey works on projects that showcase the latest hybrid cloud technologies and solutions from IBM.

Frank De Gilio is an IBM Distinguished Engineer. He works at the IBM World Wide Client Technology Centers with a global focus on client enterprise infrastructures. He is the IBM Systems Chief Architect for cloud computing. Frank's recent projects have been focused on providing enterprise-wide cloud solutions to IBM clients who are interested in using cloud computing. His unique approach looks at the holistic requirements on cloud of an enterprise, uniting the development, operational, and business aspects of the cloud deployment model to ensure that a business is considering at all of the implications when implementing the technology.

Randy Frerking is an Enterprise Technical Expert in Infrastructure and Platform Services at Walmart Technology, Walmart Stores Inc. He is responsible for designing and developing foundational enterprise and cloud services to satisfy various IT and business needs. He provides training and socialization of technological concepts, influences direction, and promotes innovation throughout Walmart's IT organization. His primary focus is on IBM z Systems engineering and software development. He has 36 years of IBM CICS and mainframe middleware experience with 8 years at Walmart Technology.

Rich Jackson is a Technical Expert in Infrastructure and Platform Services at Walmart Technology, Walmart Stores Inc. He is responsible for designing and developing foundational services to satisfy various IT and business needs. He provides training and socialization of technological concepts, influences direction, and promotes innovation throughout Walmart's IT organization. His primary focus is on z Systems, but he has participated in grid computing planning and design as well. He has been with Walmart for over 11 years with a background in storage and z/OS management.

Charlie Lawrence has 50 years of experience in various aspects of operating systems, and application design, development, and testing. He has contributed to the development of technical and end user curriculum as well as associated training materials for IBM and other technology companies. Charlie served as an IBM technical instructor and course developer for 18 years during his 30-year IBM career, with a focus on the internal logic and concepts associated with z/OS and VM. Charlie has co-authored three other IBM Redbooks.

Kellie Mathis is a Director in IBM Systems, and has been with IBM for 31 years. Kellie works with clients to understand and develop the business justification and use cases for using IBM z Systems in modern ways. She helps clients overcome old paradigms of mainframes within their organizations and develop a modern plan to use the power of z/OS systems.

The project that produced this publication was managed by **Marcela Adan**, IBM Redbooks Project Leader - International Technical Support Organization.

Thanks to the following people for their contributions to this project:

Joe Foti
IBM WW Client Center Hybrid Cloud Program Manager

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  `http://www.facebook.com/IBMRedbooks`

- Follow us on Twitter:

  `http://twitter.com/ibmredbooks`

- Look for us on LinkedIn:

  `http://www.linkedin.com/groups?home=&gid=2130806`

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  `https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm`

- Stay current on recent Redbooks publications with RSS Feeds:

  `http://www.redbooks.ibm.com/rss.html`

1

Introduction

People who have an enterprise cloud strategy win. People who don't… lose. It's as simple as that.
Frank De Gilio

For more information, see the IBM Center for Applied Insights article "Competitive advantage is in the clouds: A discussion with Frank De Gilio" at:

`http://ibmcai.com/2015/08/04/competitive-advantage-is-in-the-clouds-a-discussion-with-frank-degilio/`

This publication helps you get started with creating IBM z/OS cloud services as part of an enterprise cloud strategy. This chapter explains the contents of the book and provides an explanation of the scenario that this book covers.

This chapter includes the following sections:

- 1.1, "Real life answers from real life experiences" on page 2
- 1.2, "An overview of this publication" on page 2

1.1 Real life answers from real life experiences

Before you considered this publication, you were most likely previously exposed to a multitude of other cloud-related publications, white papers, and blogs. Those publications might have described or offered cloud concepts, and suggested implementation approaches.

Perhaps you are already convinced that you need to take a cloud-related approach, but are unable to find practical information for how to get started. This publication provides a practical approach.

Co-authors Randy Frerking and Rich Jackson both of Walmart Technology, Walmart Stores Inc. have spent the last three years developing and implementing z/OS cloud services in an enterprise production environment. This book uses their experience as an example to help you focus on making the transition from traditional IT deployment to a cloud model.

The processes, approach, and considerations that are discussed are real and relevant. Walmart has already demonstrated the effectiveness of its approach by creating services that provide the capability needed for the high demands of the holiday shopping season, for multiple consecutive years.

Embarking on a cloud strategy can be overwhelming. No shortage of approaches to cloud computing exists. This book focuses on a pragmatic approach for enterprises that are struggling to take advantage of their business assets in the cloud.

1.2 An overview of this publication

Until recently, the primary focus of cloud has been data center virtualization and the most efficient use of hardware resources. Today, cloud is focused on providing capabilities to increasingly mobile-dependent consumers. The goal is to provide relevant capabilities that use the latest technology while taking advantage of existing business assets.

Because cloud means different things to different people, each chapter of this book focuses on conversations and implementations of a particular aspect of the National Institute of Standards and Technology (NIST) definitions of cloud computing as it relates to enterprises.

Chapter 2, “Cloud from a services perspective” on page 5 defines the relevant aspects of cloud computing that are of interest when creating an enterprise-level implementation. This chapter covers NIST definitions and introduces terminology and concepts that are used in the other chapters of this book.

Each remaining chapter explains the importance of a particular NIST characteristic, the role of z/OS in accomplishing that characteristic, and how it was implemented by Walmart Technology as follows:

- Chapter 3, “Shared pool: Configurable resources” on page 25 provides an overview of resource pooling and explains how this feature is important to implementing resource sharing, which is a key characteristic of cloud environments.
- Chapter 4, “Elastic environment” on page 33 Introduces elasticity, which refers to the ability to adapt capacity to workload demands. Doing so provides the illusion of unlimited capacity to the consumers. This chapter includes considerations for planning physical resources and implementation considerations to achieve elasticity in cloud environments.
- Chapter 5, “Ubiquitous access: ReST enabling your runtime” on page 41 describes the essential cloud characteristic of broad network access (ubiquitous access) and its

relationship to web services to enable access to capabilities through the network in a cloud environment. The objective is to make services available to the broadest range of platforms and clients possible.

- Chapter 6, "Metering and measuring" on page 53 provides an overview of the concepts of metering and measuring in a cloud services environment. These metrics are key to determining the impact of demand on resources and the related ability to meet service level goals, agreements, and billing. This chapter includes practical considerations, examples, and real life experiences.
- Chapter 7, "Self-service provisioning" on page 61 explains on-demand self-service, which enables customers to provision computing capabilities such as server time and network storage without requiring human interaction with each service provider. This chapter includes examples that show how to deliver on-demand self-services by using automation and self-service portals.

2

Cloud from a services perspective

This chapter defines the basic terminology and concepts that are the foundation for the solution approach and implementation of enterprise cloud services that are described in this book.

This chapter includes the following sections:

2.1 An overview of terms

One of the complicating factors associated with cloud is just how nebulous the word *cloud* can be. For some, it represents a method for consolidating resources. For others, it is a means of accelerating development. This publication follows the formal definitions of cloud and cloud-related terminology as defined by the National Institute of Standards and Technology (NIST), which can be found online[1]. NIST is an agency of the US Department of Commerce. When people talk about cloud, they usually use terms and concepts that correspond to the characteristics defined by NIST.

A complete description of open standards and every conceivable implementation could fill volumes and would distract from the topic at hand. For the purposes of this document, accept that open standards provide a way to define and standardize methods and techniques for various approaches to cloud and applications development including the related application programming interface (API).

An introduction to open standards is provided in 2.1.3, “An overview of open standards” on page 8.

2.1.1 Cloud attributes

Staying focused and productive in the cloud space requires knowledge of and adherence to a well-established and widely accepted set of guidelines. Personal definitions or conceptions of cloud vary widely, but use the five essential characteristics of cloud, three service models, and four deployment models as defined by NIST as the foundation for your understanding moving forward.

The five essential characteristics of cloud

The five essential characteristics of cloud are introduced in this section. Each introduction contains a reference to the corresponding chapter in this book that has more details about the characteristic.

- On-demand self-service

  The *on-demand* aspect of self-service can mean different things to different people. The most useful definition of on-demand self-service is that services can be provided with no human intervention required. It means that you (as the user of some web or mobile interface) or an application (on your behalf) can do or request something that triggers an action. Likewise, that action results in the provisioning of some resource or resources. This can include resources that you are not aware of.

  For additional information about on-demand self-service, see Chapter 7, “Self-service provisioning” on page 61.

- Broad network access

  The essential cloud characteristic of broad network access (ubiquitous access) basically dictates that capabilities are accessible through *standard interfaces* over an Internet Protocol network. A key part of this description is the use of the term *standard*. Using standards for transport (as in TCP/IP and HTTP) and accessibility (as in URI and JSON) ensures that service capabilities are available to the broadest range of platforms and clients. A benefit of this model is the resultant abstraction and loose coupling of the clients from the specific systems of the service providers.

[1] The NIST Definition of Cloud Computing: Recommendations of the National Institute of Standards and Technology http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

For additional information about broad network access, see Chapter 5, “Ubiquitous access: ReST enabling your runtime” on page 41.

- Resource pooling

  Resource pooling is considered an essential characteristic of cloud computing. The resources under consideration in the pooling and subsequent allocation process include storage, processing capacity, memory, and network bandwidth and access.

  For additional information about resource pooling, see Chapter 3, “Shared pool: Configurable resources” on page 25.

- Rapid elasticity

  The NIST definition of rapid elasticity is the capability to provision and release resources quickly, and doing so automatically as the scaling requirements of your environment change. Capabilities (functions and resources) can be elastically provisioned and released to meet the needs of the consumers of those resources.

  For additional information about rapid elasticity, see Chapter 4, “Elastic environment” on page 33.

- Measured service

  Measured service is about tracking who is using what and reporting the respective information back to the involved parties. Resource usage must be monitored and reported at the appropriate level of granularity for the type of service and the intended audience.

  For additional information about measuring, see Chapter 6, “Metering and measuring” on page 53.

Three service models

The *suffix as a service* has been attached to the end of a great number of things lately. Although it is generally understood that it means that there is some sort of self-service and scalability, NIST formally recognizes only three *as a service* cloud deployment models. Each of these deployment models represents a clear level of abstraction across the IT stack. They are Infrastructure as a Service, Platform as a Service, and Software as a Service. These are abbreviated to IaaS, PaaS, and SaaS.

- Infrastructure as a Service (IaaS)

  This is the lowest level of service provided, and allows the underlying resources to be offered on demand. Resources include compute, storage, and network so you no longer have to worry about data centers, power and cooling issues, or buying hardware. Deployment examples include on-premises, IBM SoftLayer®, Amazon Web Services, and GoGrid.

- Platform as a Service (PaaS)

  Moving up in the stack, the operating system, application server, and programming language used to design and develop applications can be provided as needed. In addition, building and testing of these applications can be done quickly, which greatly improves the time to bring new function to the users. Examples of deployment options include on-premises, IBM Bluemix®, Microsoft Azure service platform, and Google App Engine.

- Software as a Service (SaaS)

  The software to host applications is provided so the provider handles only the administration of users. These applications are usually referred to as apps that are *born on the cloud.* Examples include IBM Bluemix, and Salesforce.com.

IBM z/OS plays a unique role in these service models. Underlying architectures like IBM Processor Resource/Systems Manager™ (IBM PR/SM™) are provided in the IaaS layer. The z/OS functions such as Workload Manager, IBM Parallel Sysplex®, and Sysplex Distributor provide PaaS to the user.

2.1.2 Four deployment models

The *cloud* deployment models specify four types of implementation. Much of the technology and capability that enable these implementations remains the same. The main differentiators across the deployment models are the intended consumers and interoperability.

- Private cloud

  The cloud infrastructure exists only for the exclusive use of a single organization. The cloud can have multiple consumers, such as business units.

- Community cloud

  The cloud infrastructure exists only for an exclusive set of users with shared concerns, needs, or mission. It can be operated, maintained, or owned by a third party and its location might also be off-premises.

- Public cloud

  The cloud infrastructure exists for use by the general public and exists on the premises of a cloud provider.

- Hybrid cloud

  The cloud infrastructure is a composite of two or more distinct cloud infrastructures. Each distinct cloud infrastructure remains and retains its unique attributes, but connectivity or communication across the members of the hybrid cloud is made possible. The portability of data and applications is enabled.

The focus in this book is on *hybrid cloud* with an emphasis on the creation of z/OS cloud services that are able to participate in or contribute to communication across the members of the hybrid cloud. This approach facilitates the portability of data and applications.

2.1.3 An overview of open standards

Open standards support interoperability. Interoperability is needed to connect or communicate between or across *hybrid cloud* members, providing access to your z/OS cloud services. You will encounter the following terms in discussions about open standards:

- Binary and text
- XML
- JSON
- ReST

Binary and text files

Binary and text files can be transferred or transmitted and then manipulated by using an open standard for data modeling that is known as Data Format Description Language (DFDL).

DFDL is used to define how binary data and text data should be managed or handled by an application. The types of files that are candidates for DFDL include text and binary files such as comma-separated values (`.csv`) or executable (`.exe`) files. The following topics list more sources of information:

- Open Grid Forum

  `https://www.ogf.org/ogf/doku.php/standards/dfdl/dfdl`

- Constructing message models

  https://www.ibm.com/support/knowledgecenter/SSMKHH_9.0.0/com.ibm.etools.mft.doc/bz90450_.htm
- DFDL industry standard formats

  https://www.ibm.com/support/knowledgecenter/SSMKHH_9.0.0/com.ibm.etools.mft.doc/ad09520_.htm

The Open Grid Forum is responsible for the DFDL specification. DFDL schemas are being provided by IBM for specific industry applications such as healthcare and financial services. The goal of the provided DFDL schemas is to minimize the need for companies to duplicate effort in creating additional DFDL schemas.

Extensible Markup Language versus JavaScript Object Notation

Extensible Markup Language (XML) provides a way to simultaneously specify content in both human-readable and machine-readable formats. Likewise, JavaScript Object Notation (JSON) is usually referred to as a lightweight data interchange format that is also easy for humans to read and write. Similarly, JSON is easy for machines to parse and generate.

The differences between XML and JSON become apparent when you consider the representation of data in your programming language of choice. The argument in favor of JSON for the transport of data in a cloud is that the transformation from JSON to internal data representation (and vice versa) is easier and more efficient. The examples and associated text that follows are provided for your consideration in favor of JSON.

The XML provided in Example 2-1 contains the tag `cyclistsTrainingEvent`. The tag-based structure describes an event called `cyclistsTrainingEvent`, which contains the tags `averageSpeed`, `distanceCycled`, and others. This XML structure can be used to describe and contain content for an event.

Example 2-1 XML as it might be used on a call to a service

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
 <cyclistsTrainingEvent>
   <averageSpeed>15.8</averageSpeed>
   <distanceCycled>69</distanceCycled>
   <eventDate>08/27/2015</eventDate>
   <firstName>Charlie</firstName>
   <lastName>Lawrence</lastName>
 </cyclistsTrainingEvent>
</root>
```

At first glance, the `cyclistsTrainingEvent` tag appears suitable as a human-readable and machine-readable data transfer format. However, as noted earlier, the benefits of XML-encoded data quickly diminishes when the data has to be processed or manipulated by an application.

At the point that the data has to be processed, a simpler *keyword:value* method of representing data is arguably a better choice. Most obvious in this comparison is the ease of correlation between structures in various programming languages and the structure of JSON.

Consider how you might represent or store the data provided in Example 2-1 using your preferred programming language. Your programming approach might involve a structure or object that correlates to a keyword:value approach. In that case, consider that a keyword:value approach to represent data can easily map to the JSON shown in Example 2-2.

Example 2-2 JSON as it might be used on a call to a service

```
If there were a service that would store information about a bicyclist's daily
ride, this might be a reasonable format for the JSON to be used when sending an
HTTP POST to the following URI:
http://services.asamplename.com/CyclingRecords

{
"cyclistsTrainingEvent": {
"firstName": "Charlie",
"lastName": "Lawrence",
"distanceCycled": "69",
"averageSpeed": "15.8",
"eventDate": "08/27/2015"
}
}
```

As an additional comparison of XML and JSON, consider how the response from a called service might be described by using XML. Such a response might be represented by using XML as shown in Example 2-3, where the tag-based structure describes `cyclistsTrainingEventResponse`. `cyclistsTrainingEventResponse` contains an `eventId` and a `timestamp`.

Example 2-3 XML as it might be used to describe a response from a called service.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
   <cyclistsTrainingEventResponse>
      <eventId>55996345</eventId>
      <timestamp>2015-08-28T11:26:55.58Z</timestamp>
   </cyclistsTrainingEventResponse>
</root>
```

Compare the XML in Example 2-3 to the simplicity of a JSON response as shown in Example 2-4. Again, the JSON is a more natural way to programmatically express information to be communicated to and from a z/OS called service.

Example 2-4 JSON as it might be used as a response from a called service.

```
The response body could look like this:
{
"cyclistsTrainingEventResponse": {
"timestamp": "2015-08-28T11:26:55.58Z",
"eventId": “55996345”
}
}
```

After the call to an application (HTTP POST) associated with the JSON shown in Example 2-2 on page 10, with a response in the form shown in Example 2-4, the `eventId` of `55996345` can be used on a subsequent call to a related service. Such a call might use HTTP GET to retrieve the contents of the stored item. On that call, the `eventId` identifies the item of interest.

2.2 A changing business model

The tradition used to be that IT handled all computing needs of a business. Programs that ran in a data center did so in an environment that enforced processes. Jobs and their steps defined the order and conditions of program invocation and their associated resources. This approach ensured that a set of business assets (including programs) formed a complete and acceptable value chain. Often this order and condition of program invocation was perceived by users as a single application that consisted of menus that could be used only in a specific order. No single object of that type of application could be accessed or used on an individual basis.

For years, this tradition has successfully kept pace with the evolving functional needs of an enterprise. This era of dependency on IT saw the rise of a culture and set of processes that are adopted to support security, availability, and business resiliency, all of which fell under the control of IT.

However, the business model for enterprise IT must change. A new and evolving business model for those enterprises wanting to survive in a new world of cloud requires the breaking apart of programs and data assets.

In the context of a changing business model, cloud, and the breaking apart of programs and data assets, lines of business now have options that were not available in the traditional IT model. One option is to turn to cloud providers for solutions that provide the wanted functions.

However, the decision to turn to external cloud solution providers does have a potential downside. When non-IT internal organizations turn to external cloud solution providers, it is often done without the advantage of experience or awareness of nuances in the services that they select, and without considering the consequences of the choices being made.

At the root of this downside is a lack of focus. Lines of business focus on functional issues. In contrast, operational issues and concerns that IT departments normally consider to protect the business might not be in scope of non-IT groups or lines of business because they are not involved in the details of running a data center.

The potential ramifications of going outside traditional IT are related to the answers or perhaps lack of answers to these questions:

- What happens if the data that goes out into the cloud is lost, hacked, or stolen?
- What happens if the government closes down the cloud service provider and the data is impounded?
- What happens if the cloud business fails?
- What happens if the service goes down. Even worse, what happens to a retailer if that downtime occurs on the busiest shopping day of the year?
- What happens if the cloud provider inadvertently compromises your security or compliance requirements?

IT has been dealing with these concerns since the beginning of the industry.

Clearly, IT should be part of the lines-of-business computing purchases and decisions, including cloud-related decisions. IT has the experience and focus to support the business as a whole because IT provides for the needs of these business units. However, IT often has a reputation of standing in the way of collaboration and moving into new technology quickly. As a result, the culture and processes that have served businesses for years can become inhibitors to the collaboration that is needed to move faster.

Businesses still need their IT staff. IT should be considered and respected as protectors of the data and the technology that uses and accesses it. But if IT is to survive and support the business needs, it must evolve and adapt to collaborate and cultivate. If they do not, the parent enterprises can find themselves falling behind the times and losing business as a result.

IT must embrace cloud. It can do so by becoming a broker for service providers and a provider of services. IT should know how to connect lines of business to service providers. By doing so, it can provide the needed functions in a timely manner while continuing to support the traditional business needs as a whole.

You must decide which activities should continue to be performed in-house and which ones need to be outsourced to cloud providers. IT needs to have good business justification for its actions. In short, IT must become more business-oriented and dynamic. This is not to say it must be a money-maker, but it must act with a business focus that is leaner and more dynamic than it was in the past. At the same time, IT must not compromise its core value to the enterprise.

In addition to brokering connections to external cloud services, IT continues to provide new services. Some functions can only be provided internally. The internal attribute or requirement might be a result of the need to retain control for security or regulatory reasons, or because the internal code provides the company's competitive advantage.

This is where the concepts presented in this book come into play. The identification of needed services as *z/OS cloud services* must be brokered between the lines-of-business and the services they require in a collaborative manner. For collaboration to succeed, IT must become lean, agile, and cost-competitive.

2.3 The hybrid cloud service model

Businesses are more focused than ever on creating applications that bring them closer to both their providers and consumers. The ability to employ the latest technologies quickly allows a business to differentiate itself from its competitors. Companies that can develop new applications quickly, while using the technologies and processes that have given them their business advantage will be the winners.

Additionally, enterprises that have provided this capability to their own lines-of-business, and have also discovered how to sell this capability to other businesses, remake their IT into a new source of revenue. *Hybrid cloud* as a service delivery model can be the enabler for such a dramatic change in the business. Before discussing the model as a whole, be sure to understand several basic concepts:

- Modularization of application components

  This concept involves an analysis of the applications currently running on your z Systems. The intent of this analysis is not to just add a ReSTful wrapper around the application. Rather, the intent of this analysis is to identify and describe basic building blocks. These basic building blocks are candidates to be offered or accessed as services. More details are in 2.3.1, "Modularization of application components" on page 13.

  The potential result is that new solutions can be created without the need to rewrite or reengineer existing applications. These solutions can use some number of those basic services in various forms such as a mobile app. In this environment, a mobile app can start a series of events, hidden from the user, that occur through a path within the hybrid cloud. The mobile app receives results that are provided by the service (or services). The app then presents the results to the user. The exchange of information is accomplished in a secure and reliable manner.

Additional nuances of solutions that use services in various forms can customize a user query in the context of where they are currently located. For example, asking an app to *find a good place for dinner* can provide responses that differ based on such factors as GPS data without even asking where you are located.

- Clearly defined service definitions

  Services are generally focused on providing some function. In addition to function, a service provider must define operational characteristics that are appropriate for the service. These characteristics define the security, availability, performance, and other non-functional aspects of the service. A specific service function can have multiple service definitions that identify different ways that it can run.

  For example, a service function might allow you to store and retrieve data. That same service function can be referenced through different definitions. Each service definition (of a service function) determines whether the data is encrypted, available all of the time, and how quickly it can be stored or retrieved. More details are in 2.3.2, “Creating services” on page 14.

- The hybrid cloud service delivery model

  This model is about connecting multiple clouds together to provide some capability. The different environments that are connected or brought together in the hybrid cloud service delivery model each have different strengths and value. As a result of bringing them together, the value of the created deliverable is greater. More details are in 2.3.3, “The hybrid cloud service delivery model” on page 15.

2.3.1 Modularization of application components

Businesses must find a way to help application developers more easily use traditional IT resources without the developers needing to understand why or how these resources work. Business capabilities (or services) must be easy to use and composable in ways never before considered or possible. When ease-of-use and composability become a reality, application developers can create solutions quickly, allowing the business to be more flexible.

Additionally, the focus of application development moves from a technical focus to a business focus. This shift of focus does not remove the need for a technical focus. Instead, the technical focus is on creating services that can be connected together in applications.

Through the modularization of application components, you can create abstractions of a transaction or set of transactions as a set of services. If done correctly, these services become the basic building blocks of future applications. These blocks can be called, stacked, and amalgamated as needed, with ease.

A key step in preparing services to be made available to cloud entities is the process of identifying and documenting clearly defined boundaries or scope of action (impact) in the context of specific inputs and outputs. That process can become an iterative effort with the goal of drilling down to the most basic business operation that can be performed.

During this process, look at service candidate and ask how this service decides whether it should do A or B. An answer might likely be to split the service into two services, one of which does A and the other does B. With that approach, both A and B have no decisions to make. Instead, the resulting services accept the input (*do what they do*) and return the output.

The potential result is that new solutions can be created without the need to rewrite or reengineer the transactions completely.

Embracing this change allows you to free a set of business functions that are currently locked in the data center. Rather than being hampered by decades of legacy applications, you can now benefit from them. You can become a service provider with cloud-enabled services. Those services can be used by a wider audience than ever before. Having access to a wider audience can also translate into income, if you market or sell those services while still protecting and maintaining ownership of the data. As a result, your data center can be transformed from a cost center to a profit center.

In addition, when you move an enterprise application that was originally a stand-alone CICS transaction to being fully cloud-enabled as a service, you gain more flexibility while encouraging innovation in many areas. Those areas are collectively referred to as *CAMSS*: *Cloud, Analytics, Mobile, Social, and Security*. This approach is not just "thinking out of the box," it is thinking out of the application.

2.3.2 Creating services

The definitions of input and output of a business function are key to opening up your service to the cloud. The defined attributes are required to support calls or invocation of the service. This requirement is related to how requests for services are transported with associated data or payloads.

Potential users of a mobile app are probably ignorant of the service that you provide. That ignorance is acceptable because the mobile app provides the users' experience by communicating with the service or an agent of the service on their behalf to present some meaningful output based on user input.

Consider the non-technical service delivery scenario of getting your morning coffee. It involves these steps:

- You place your order for a cup by saying "large black coffee to go please."
- Your server gets a cup.
- Your server pours coffee in the cup.
- Your server places a top on the cup.
- Your server hands the cup to you.

The only set of tasks you might perceive is the *get-pour-place-handoff*. You most likely care only about the *handoff*. If this is a drive-through order, you see *only* the *handoff*. Being ignorant about which services participated in the presentation of the cup of coffee to you is fine. All that you really care about is the presentation of the coffee.

However, many tasks must occur for you to successfully request and receive your morning coffee in the form of large-black-to-go cup. Someone had to pick the beans, pack and ship the beans, roast the beans, deliver the beans, grind the beans, place the ground beans in a basket, and so on.

In the context of your large-black-to-go order, if you were using a mobile application, you need a set of corresponding services such as *pick and return 47 coffee beans* (assuming that 47 coffee beans is the number required to make a good tall black bold cup of coffee). Picking and returning coffee beans might be the *most basic service* that you identified during the modularization process mentioned previously. Defining this service in terms of what it does, what it needs as input to perform the service, what impact it has, such as possible depletion of coffee beans, and what it provides as output can help you resolve the other issues. Those issues include how this service is called and how it returns (transports) output.

The issues and details associated with calling a service and returning results do not matter unless you begin with the basic steps of defining and refining your services. Moving to a hybrid cloud is a journey and does not happen instantly. But it can happen through a series of evolutionary, well planned and orchestrated steps that begin with the definition of services.

An important point is that the correct level of abstraction depends on the type of services that a company wants to provide. Abstractions that are too low level can affect the performance of the applications that use the services. Abstractions that are too high level can also be too slow to provide the exact information that an application needs while supplying unneeded information. The key to providing the correct level of abstractions is defining the appropriate use cases for the service. A benefit of creating services by using this approach is that you can always divide a service into components later if necessary. Because each service is an abstraction to the caller, such changes to the underlying service do not impact the caller if the API remains consistent.

2.3.3 The hybrid cloud service delivery model

The hybrid cloud can be viewed as a service delivery model that enables innovation in the rapid delivery of services. The manner in which the services are provided simplifies the task of bundling or packaging to satisfy the ever-growing demand for new applications. The resulting applications can be based on services from multiple sources, not just your services. The result might be perceived by users as a mobile app without regard for, concern for, or interest in what makes it work. All that they see are the results of the application.

Figure 2-1 shows the hybrid cloud as a service delivery model.

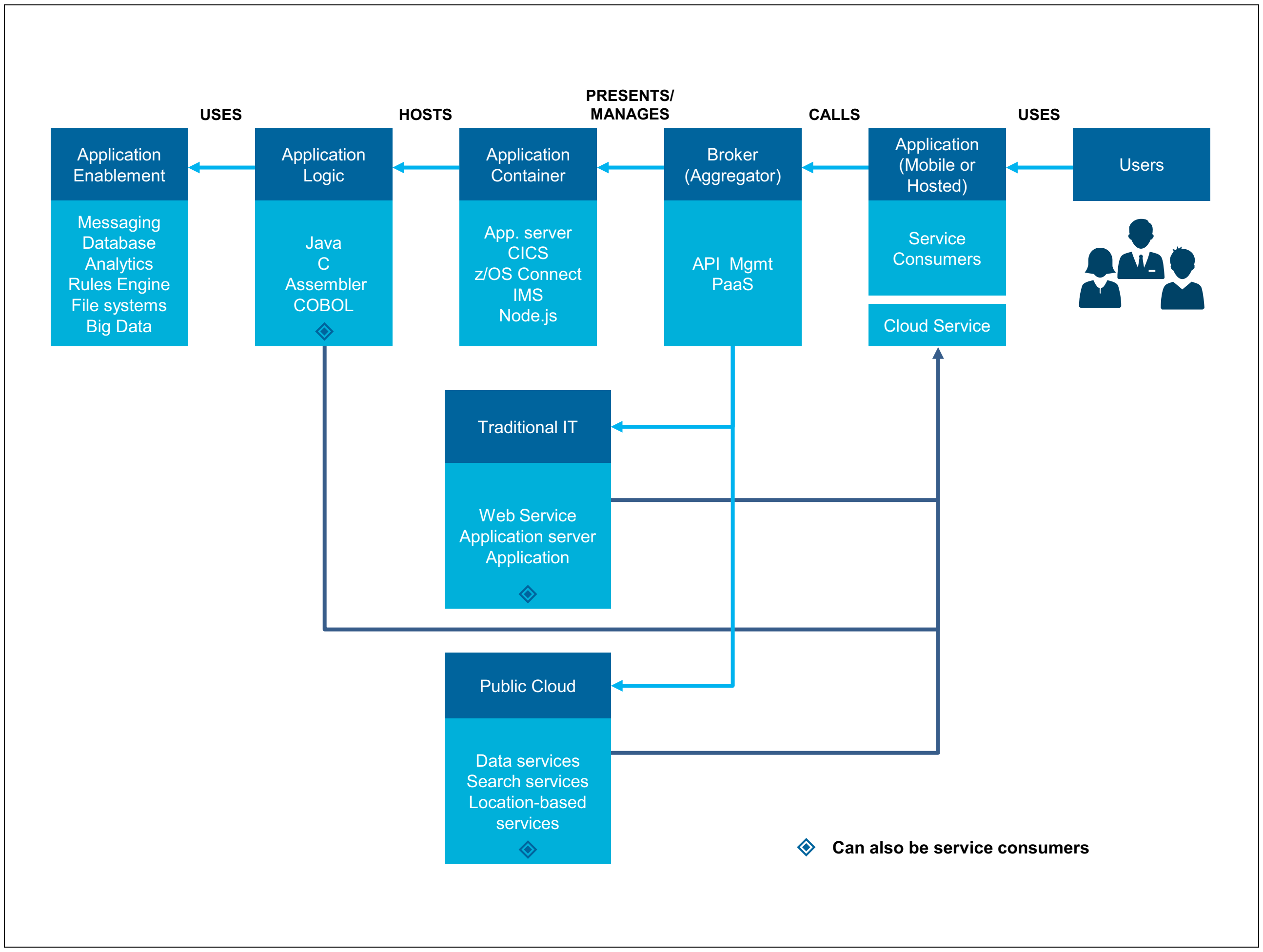

USES
HOSTS
PRESENTS/ MANAGES
CALLS
USES
Application Enablement
Messaging Database Analytics Rules Engine File systems Big Data
Application Logic
Java C Assembler COBOL
Application Container
App. server CICS z/OS Connect IMS Node.js
Broker (Aggregator)
API Mgmt PaaS
Application (Mobile or Hosted)
Service Consumers
Cloud Service
Users
Traditional IT
Web Service Application server Application
Public Cloud
Data services Search services Location-based services
Can also be service consumers

Figure 2-1 The hybrid cloud service delivery model

2.3.4 The application logic

A cloud-enabled application starts like any other application. It can be written in Java, C, assembly language, COBOL, or other programming language running on any platform. The application logic that is circled in Figure 2-2 on page 17 provides the logic for a specific set of functions. The application can depend on or be enabled by middleware software. The application enablement software provides messaging, database, analytics, rules engine, file systems, and big data support and functions. This pre-cloud application environment might be familiar to you. You might have invested and reinvested in the development of such applications over the years, and do not want to waste that investment.

The original way of running a program in this application environment includes logging on to TSO and submitting a job.

Figure 2-2 shows the creation of the application.

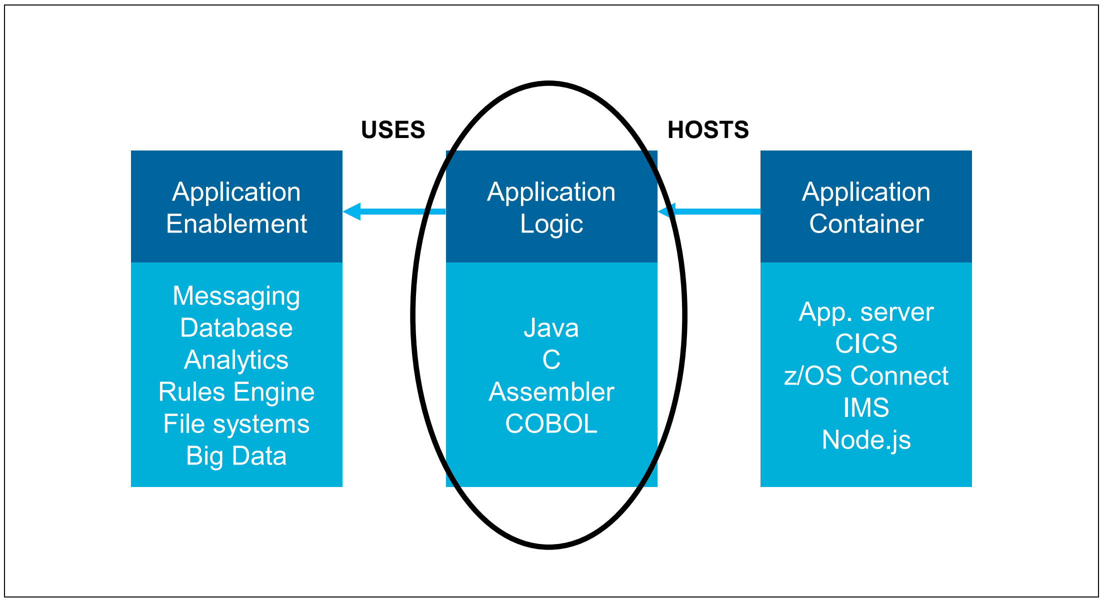


Figure 2-2 Start of an application

As the world and IT changed, so did the need for applications to exist beyond a command line or a submitted job. Most people expect that applications are available in web environments. This expectation has led to the requirement for both hosting and transport solutions. With application containers, as shown in Figure 2-3, application logic can be made accessible through an application server, CICS, z/OS Connect, IBM IMS™, or Node.js.

The three blocks (*application enablement*, *application logic*, and *application container*) can be viewed as an encapsulation of an application in a way that provides a simple set of inputs and outputs. Concealing the complex details of the implementation from users is the purpose of the API. The API is the way to invoke and communicate with applications.

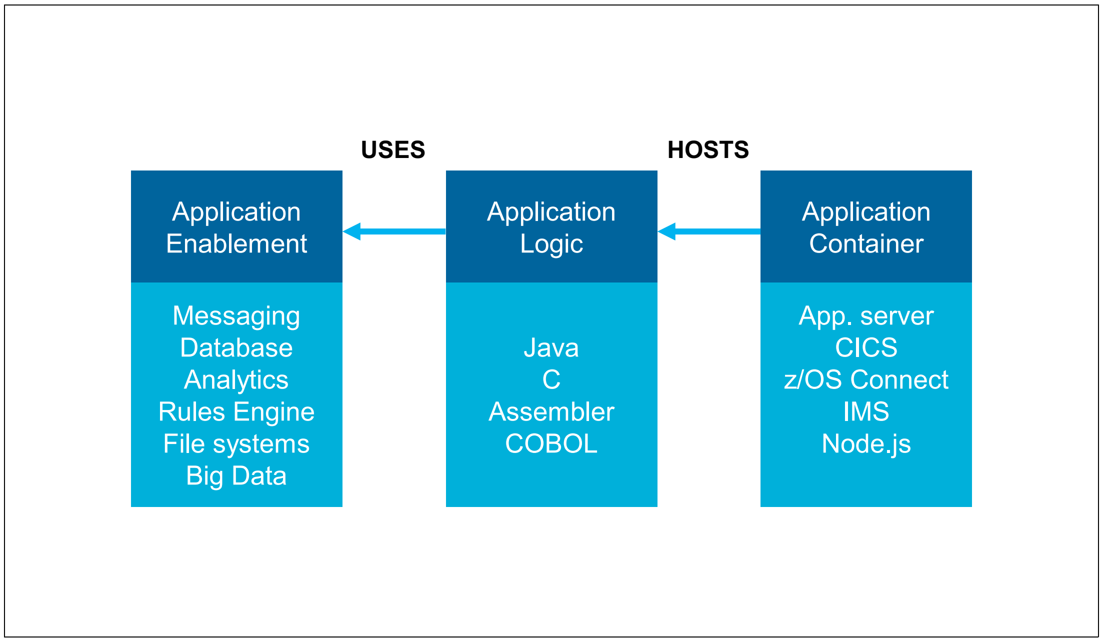


Figure 2-3 Application logic accessible through an application container

2.3.5 Broker aggregator

Expanding the view of opening an application to consumers, where the consumers are consumers of the application rather than consumers in a business-to-consumer (B2C) sense, the role of a broker becomes important. The role of the broker includes presenting and managing requests to the application container. The broker can also serve as an aggregator of applications and their services. The broker can also serve in a metrics and metering role.

The relationship of the broker to the application portion of the service delivery model is shown in Figure 2-4. API management and PaaS facilitate the deployment of applications quickly and easily.

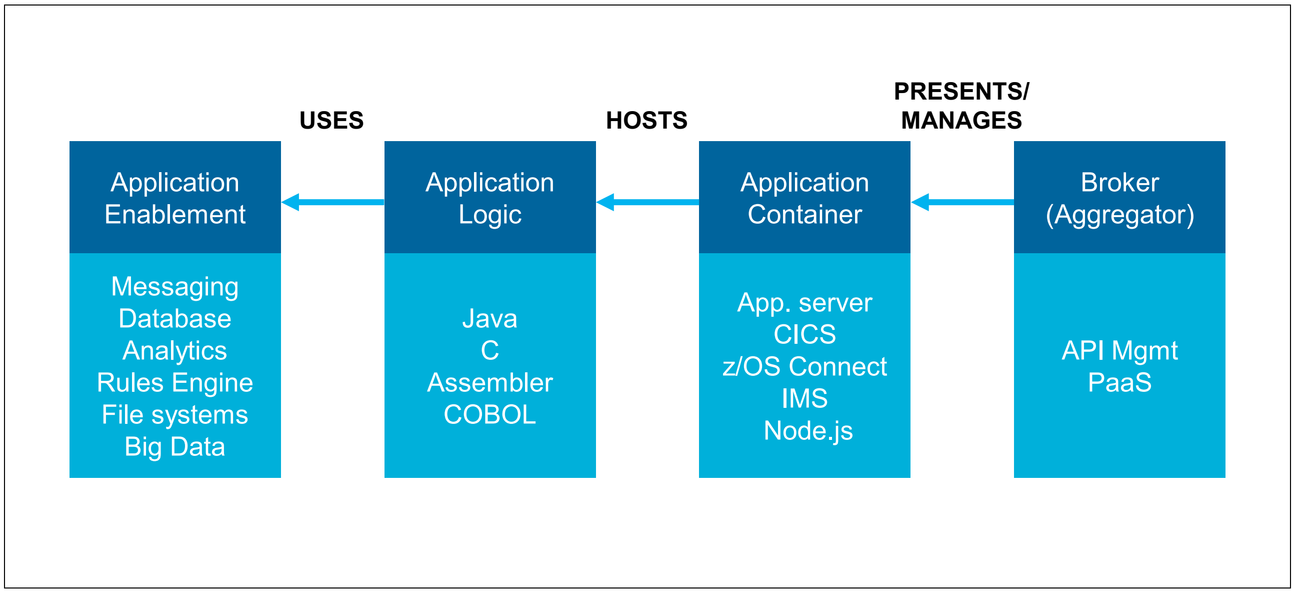


Figure 2-4 Broker (aggregator) in relation to the application container

Notes: In the context of the broker (aggregator) relationship to the application container as shown in Figure 2-4, note the following considerations:

- The goal is that your application becomes the path that the consumer of the service wants (the “desire” path). Before the service becomes a *desire path* to the consumer, the service must be the desire path of the application developers.
- When application developers discover that using a service is easy, that service will most likely become part of an application.
- It is likely that the consumer of the application has no knowledge or interest in the actual service used by the application.

The broker (aggregator) shown in Figure 2-4 provides a way to manage the API in an enterprise manner. The broker does not provide a level of service. Instead, the broker identifies the level of service that an API implementation will provide for the consumer. In this context, the broker (aggregator) also provides transactional accountability that includes tightly managed user access, usage tracking, and conversion, transformation, or substitution where needed.

With a broker (aggregator) in place, the temptation for developers to avoid in-house IT *processes* to access applications is minimized. In contrast, a non-brokered environment can be difficult to manage and track, contributing to problems that might outweigh any possible benefit. Non-brokered environments often encourage avoiding in-house IT processes, which can result in the creation of what is referred to as *shadow IT.*

The broker (aggregator) provides a catalog of available services and presents them to potential consumers. The additional benefit of using a broker (aggregator) is that you are not necessarily limited to just your services when building a catalog. You can also take advantage of public cloud offerings from third-party providers that fit into the aggregation that you are creating.

In the context of access, users are consumers of applications. The applications, whether mobile or hosted, act as consumers of services that are accessed by calls satisfied through a broker (aggregator) as shown in Figure 2-5.

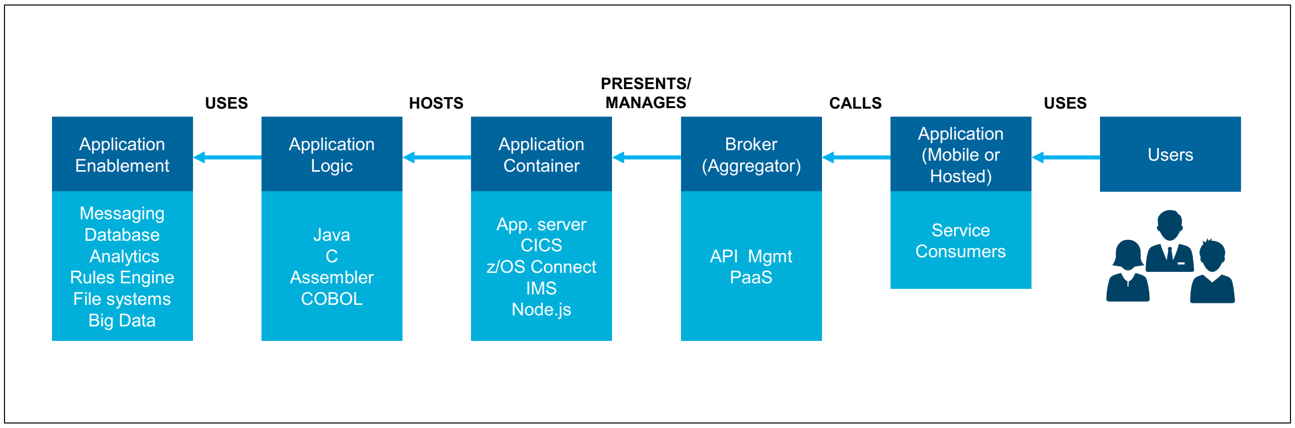


Figure 2-5 The user in relationship to the service delivery model

The details and form of the request for services can vary depending on who is starting the request. For example, suppose a user (through a hosted or mobile app) uses the following line to call upon a service:

```
example.com/inventory/product/12345
```

The service might actually be called through an intermediary using something like the following line:

```
example.com/services/corp/applications/v5/product/12345
```

The details and complexity of your infrastructure can be abstracted to the level needed or appropriate for the user-facing application.

You are not limited to only your company’s services when building a catalog. You can also take advantage of public cloud offerings from third-party providers. The resulting collection of catalog services (the company’s own services and services from third-party providers) can be used to create powerful applications.

Figure 2-6 demonstrates an application that calls services that are made available through the broker and third-party cloud services. Third-party cloud services can include web services provided through traditional IT as well as data services, search services, and location-based services.

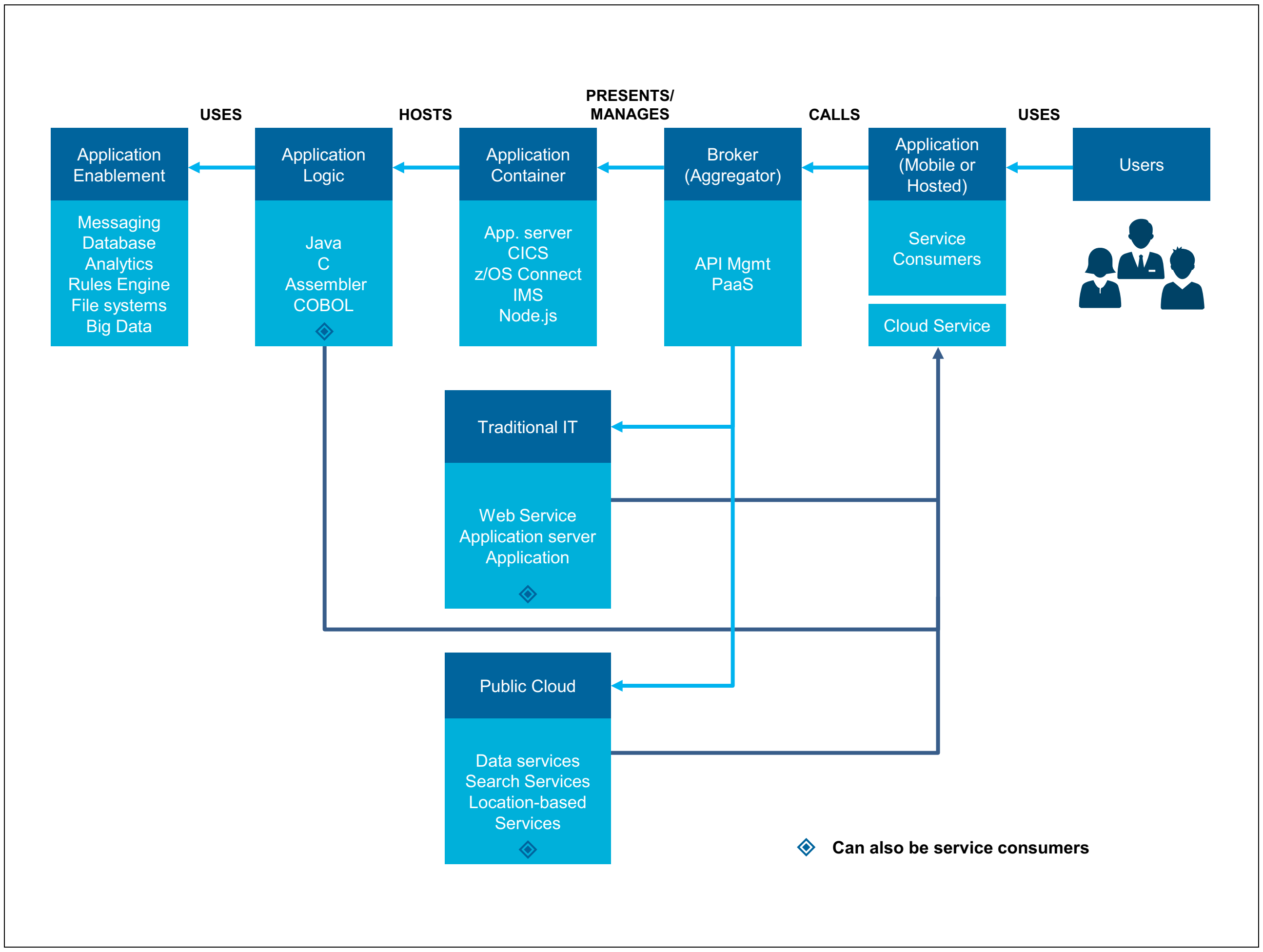


Figure 2-6 The big picture

Keep in mind that traditional IT offerings (as shown in Figure 2-6) do not need to be fully modular. However, they must to be ReST consumable and added to the catalog of the broker (aggregator) as an asset.

Consider the following two scenarios in the context of what might happen when something is requested or started by the user.

Scenario 1: A mobile app user wants to find a local coffee shop

This scenario includes the following steps:

1. The user's mobile app makes a series of calls to services. These services are most likely identified and created through the process described in 2.3.1, "Modularization of application components" on page 13.
2. The calls to services rely on API management, part of the broker (aggregator) in Figure 2-6 on page 20.
3. The application container serves as the host of the application logic, which is accessed through an application server, CICS, z/OS Connect, IMS, or Node.js.

4. The JAVA code relies on the function represented in Figure 2-6 on page 20 as application enablement such as messaging, database, analytics, rules engine, file systems, and big data support and functions.
5. The located information is returned to the mobile app.
6. Before or while formatting and presenting the returned information to the user, the mobile app can also make several public cloud calls to data, search, and location services to gather directions, traffic, and weather conditions. The information gathered by using the public cloud can also be formatted and presented to the user in the context of directions and ratings.

Scenario 2: A mobile app user wants to make a payment on a credit card

This scenario includes the following steps:

1. The user's mobile app prompts the user for account information and payment amount, then makes a series of calls to the core application.
2. The application container, perhaps through CICS web services, communicates with a COBOL application.
3. The COBOL application relies on a function that is provided by or through security software that checks authorization.
4. The payment is processed, the receipt is emailed, and the results are returned through the service delivery model, resulting in a thank you message on the user's device.

Making a payment on a credit card is an example of a transaction that depends on a mature and secure enterprise infrastructure. The needed and expected integrity of this transaction versus a seemingly less important act of *liking* a picture on Instagram or finding a local coffee shop should be obvious. However, the reality is that the mechanism for making the call to perform a secure payment transaction does not have to be much more complicated than "liking" a picture. The hybrid cloud as a service delivery model provides a *friendly* and *convenient* environment for all varieties of applications that are created using z/OS services.

2.4 Simply an endpoint

"To the application developer, it is simply an endpoint"

Randy Frerking: Walmart Technology, Walmart Stores Inc.

Keeping the focus on the consumer of a service represented by the application (mobile or hosted) component in Figure 2-7, it becomes clear that the users of the application do not need to know anything about or even be aware of the services consumed by the application.

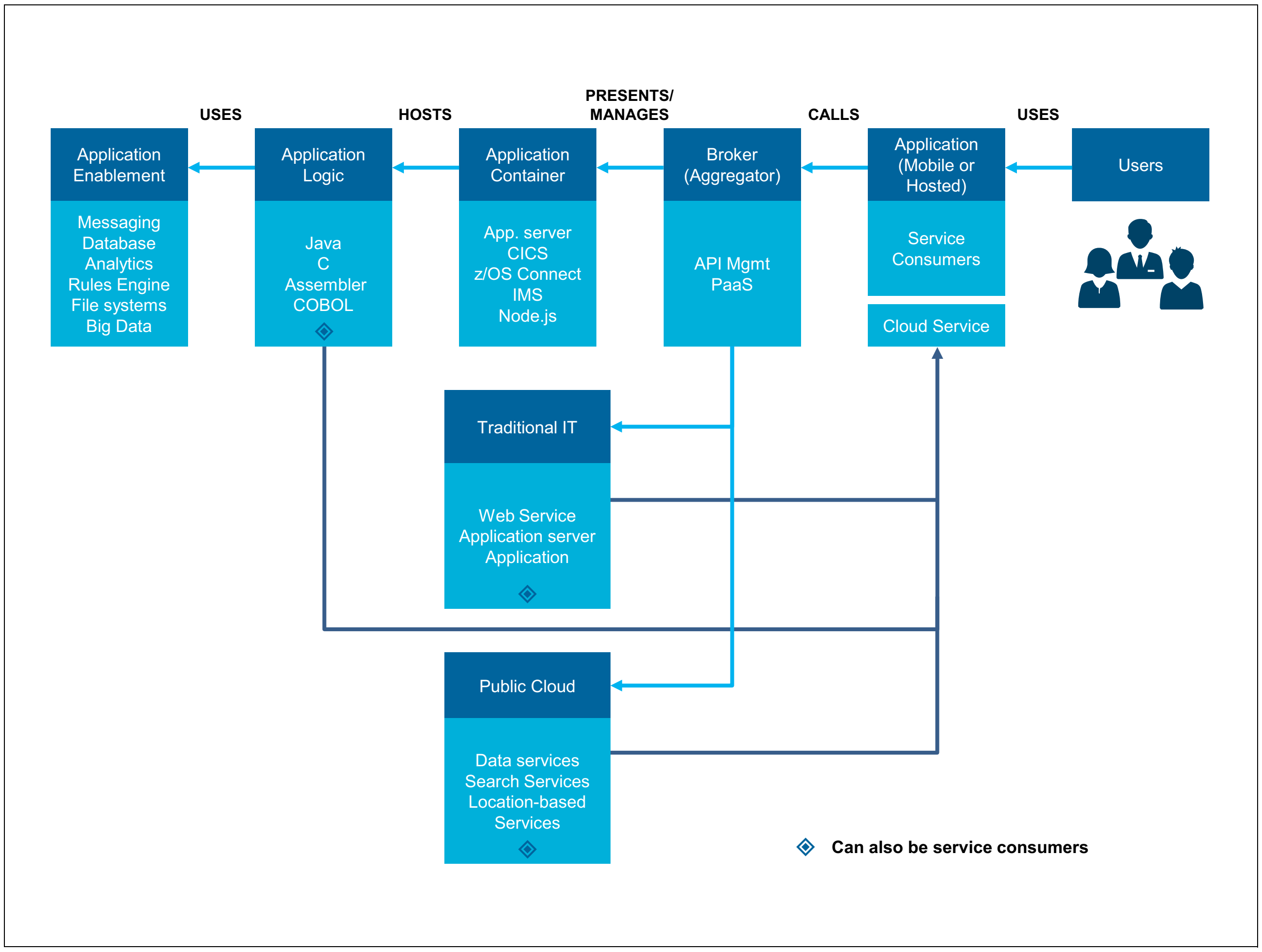


Figure 2-7 Focus on the consumer of the service

The developer (or composer) using the broker (aggregator) to create an application only needs to know the access point to call to use a service. In "Scenario 2: A mobile app user wants to make a payment on a credit card" on page 21, the developer of the credit card payment application might have requirements that call for a secure, reliable, and low-cost service. How the service provider provides that service might not be important. In fact, the service is simply an endpoint that meets the application's requirements.

As stated in 2.3.5, "Broker aggregator" on page 18, services must be easy to use. If they are not easy to use, then application developers are not likely to use them as part of an application. Likewise, an easy to use service that meets the requirements of the application being composed has a greater chance of being selected from the list of assets available in the broker (aggregator).

The overall benefit to application developers is that z/OS cloud services, in addition to meeting the requirements of the application, are able to deliver all of the characteristics of a z/OS environment behind a layer of abstraction. The application developer does not need to have any knowledge of the parallel sysplex, LPARs, address spaces, or even the role or existence of z/OS.

2.5 Services: The evolution of SOA

Services that are defined in this book will be familiar to developers who have been building on service-oriented architecture (SOA). In many ways, *cloud services* are an evolution of SOA. Cloud services follow an API model defined in SOA. Cloud services rely on HTTP for communication, as do many SOA applications. Both are based on a loosely coupled environment.

Those who invested in SOA depend on SOAP as the transport protocol. Although SOAP is often used over HTTP, it can also flow over SMTP or other protocols like IBM MQ. SOAP has a set of rules that define the communication, operation, and location of a service. Web Service Definition Language (WSDL) defines these rules. The SOAP approach provides a rich capability that supports complex interactions that allow state and context to be maintained between client and server.

In contrast to applications developed for a SOAP environment, cloud services are typically implemented with ReST, which is a much lighter weight interface with less supporting infrastructure. ReST has these characteristics:

- Unlike SOAP, ReST runs over only HTTP and does not support context or state management.
- Unlike SOAP, the rules for ReST are much less stringent. SOAP requires XML. ReST often uses JSON, but does not require it.

Cloud services are a method for implementing a lightweight SOA infrastructure. Because ReST does not have all of the structure to support state or context, it can perform quickly with much less infrastructure. Its lightweight nature makes it an excellent option for implementing capabilities quickly. Because agility and speed are at the heart of the drive for cloud implementations, ReST-enabled cloud services are a perfect fit for businesses that are looking to provide new functions quickly.

2.6 Mapping cloud concepts to z/OS capabilities

Many *cloud* teams do not normally automatically associate or connect *cloud* with z/OS capabilities because those teams lack z/OS backgrounds or have not worked with z/OS centric teams. However, it is easy to describe how z/OS provides cloud functionality.

Figure 2-8 depicts a high-level academic representation of a z/OS system (single LPAR image). The z/OS system representation includes required components in relation to all the NIST essential characteristics of cloud.

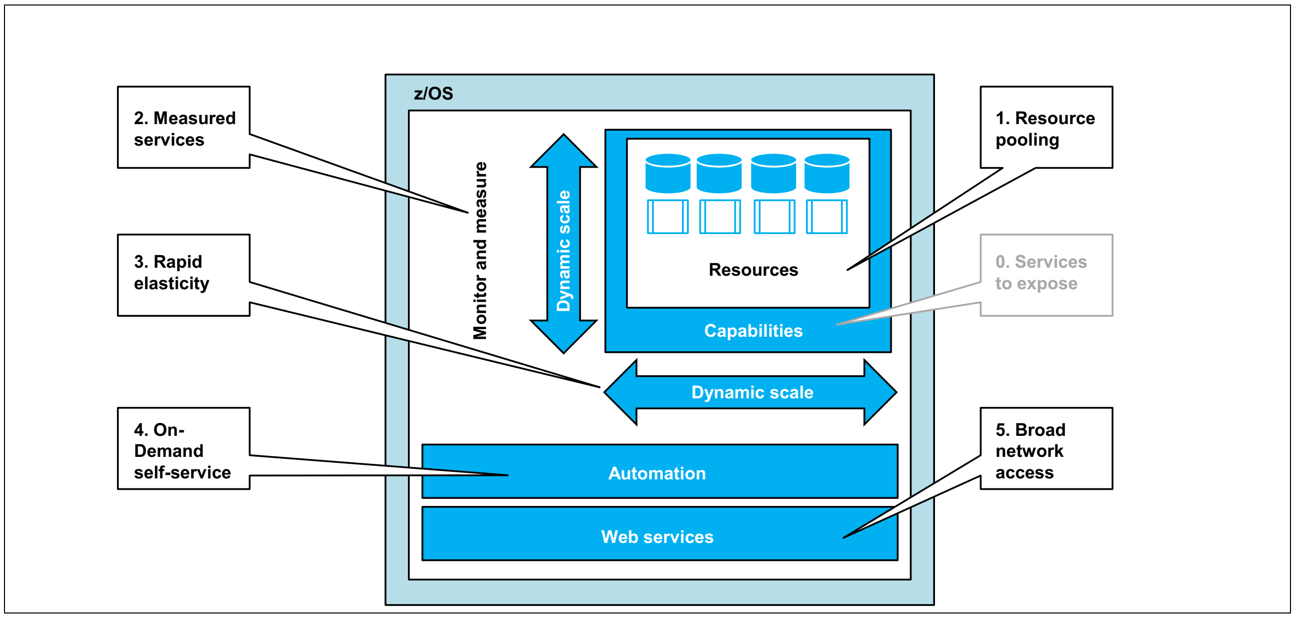


Figure 2-8 Representation of a z/OS system (single LPAR image)

Figure 2-8 provides a summary depiction of a cloud delivery approach in a z/OS context. The numbered items (1 - 5) in concert provide a capability (#0) as a service. This capability or service is composed of pieces pulled from various pools (#1) of system resources. The utilization of the resources is continually measured and tracked (#2) using system components such as System Management Facilities (SMF). Resource assignments for the service also expand and contract (#3) to meet demand. Each instance of the service is automatically composed and supplied to the user at their self-initiated request (#4). Both the acquisition and subsequent consumption of the instantiated service are achieved by using web services (#5) to ensure the broadest accessibility.

The chapters that follow explain each of these cloud characteristics in more detail in the context of related z/OS system components that are used to provide them.

3 Shared pool: Configurable resources

Resource pooling. *The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, and network bandwidth.*

The NIST Definition of Cloud Computing
http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

Resource pooling is considered an essential characteristic of cloud computing. The resources that come under consideration in the pooling and subsequent allocation process include storage, processing capacity, memory, and network bandwidth and access.

This chapter includes the following sections:

3.1 Resource pooling

Because resource pooling is an inherent characteristic of the IBM z/OS platform, minimal effort is required to enable some amount of it. Virtually any processing on z/OS, even existing legacy processes, already uses pooled resources to an extent.

Virtual storage is a prime example of a resource pool inherent to z/OS. It gives a tenant running within the environment the perception that they have access to memory throughout the addressable range. Each tenant has the same perception, due to the actual resources of central and auxiliary storage being managed by the system behind a layer of abstraction. Other examples of these inherent resource pools include resources like processor cycles and storage groups. See Figure 3-1.

Even beyond the resource pooling that is provided as part of the platform itself, other forms of resource pooling can be configured and used for service delivery, as an example, high availability (HA) CICS regions.

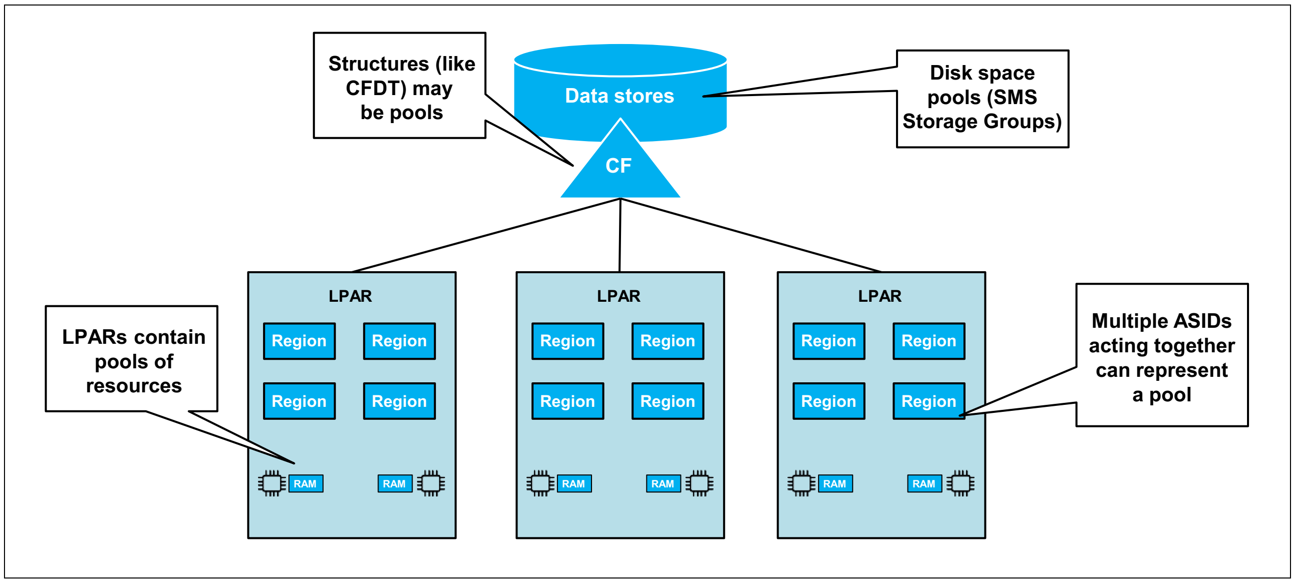

Structures (like CFDT) may be pools
Data stores
Disk space pools (SMS Storage Groups)
CF
LPAR
Region
RAM
LPARs contain pools of resources
Multiple ASIDs acting together can represent a pool

Figure 3-1 z/OS resource pools

The use of parallel sysplex can greatly expand the available resources of a hosting environment by creating *pools of pools* for some resources. Likewise, larger broader pools of other resources are distributed across the sysplex as depicted in Figure 3-2.

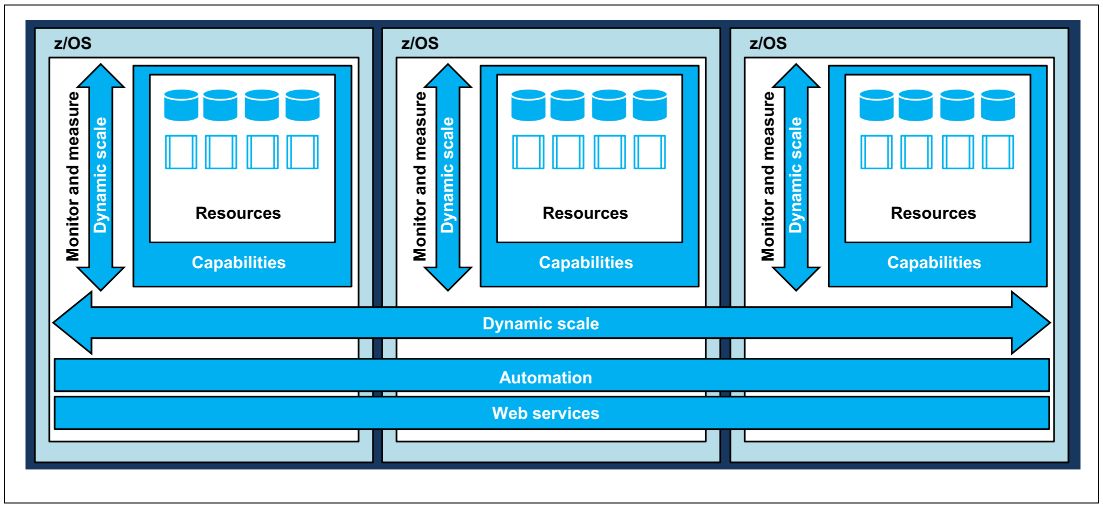


Figure 3-2 Pools shared across parallel sysplex

A combination of automated processes and controls are necessary to ensure that appropriate assignment and distribution of the resources is accomplished. Unlike in z/OS, in most of the currently common distributed cloud models, additional products that complement the operating systems and hypervisors are necessary to achieve this capability.

With z/OS, these components are largely provided by the operating system itself. The assignment and reassignment of resources in an abstracted fashion is performed by the operating system. Likewise, numerous constructs can provide controls on how the resources are managed.

These controls include workload management (WLM) policies, service classes, and Data Facility Storage Management Subsystem (DFSMS). DFSMS is often referred to as SMS. These controls facilitate workload priorities.

As shown in Figure 3-3, *Storage Group* represents a pool of storage volumes. These DFSMS managed volumes are where the service associated data sets are located. This storage group enables support for bursts of transactions for unpredictable and seasonal volumes.

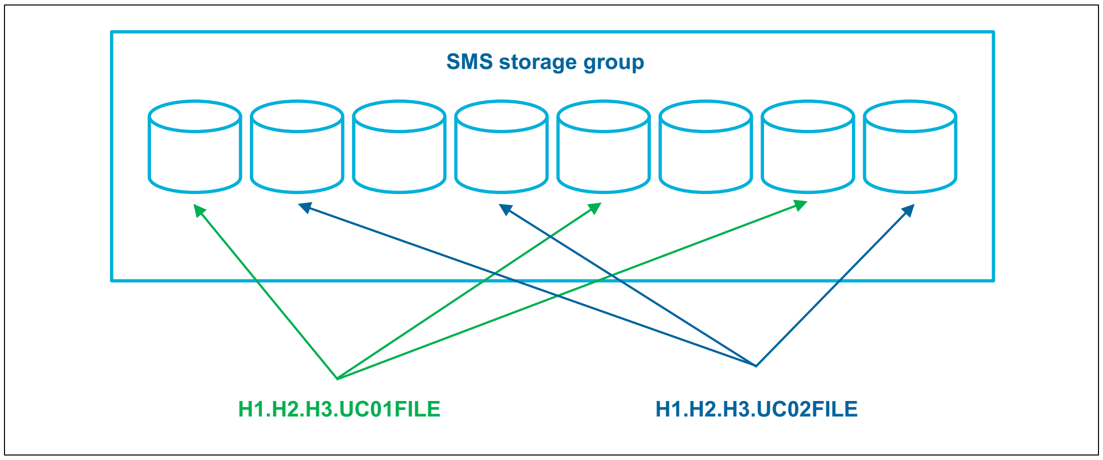


Figure 3-3 SMS storage groups

SMS constructs can be used to control disk storage allocations and performance characteristics. However, extra mechanisms might be needed to accomplish the required level of resource management. You might discover that most general needs are covered by default, whereas specific custom needs must be addressed individually.

As shown in Figure 3-4, clusters are groups of regions that act together. Each address space identifier (ASID) or region in the group share underlying resources, and satisfy the same workloads.

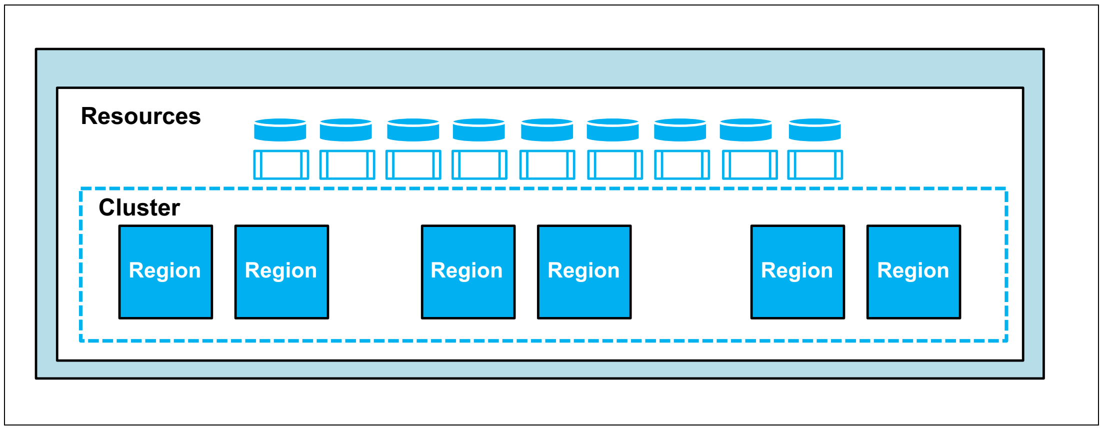


Figure 3-4 Clusters of regions

To share workloads, all logical components of a *service* must be defined to each region in the cluster. Figure 3-5 shows the logical components of a service instance. Note the use of UC01 in each resource name. All related constructs in Figure 3-5 within the group UC01 contain UC01 as part of their identifier.

```
EX G(UC01)
ENTER COMMANDS
 NAME       TYPE          GROUP
 UC01VSAM   FILE          UC01
 TCLUC01    TRANCLASS     UC01
 UC01       TRANSACTION   UC01
 UC01       URIMAP        UC01
```

Figure 3-5 The logical components of a service instance

Tenant isolation is provided within this cluster through application of the naming convention that requires the group name to be part of the resource name. Additional instances of the *service* can be defined to the cluster by using a different identifier. Figure 3-6 shows a group named UC02 with all logical constructs having UC02 as part of the resource identifier names. This naming convention is relevant to many aspects of this approach to service delivery.

```
EX G(UC02)
ENTER COMMANDS
 NAME       TYPE          GROUP
 UC02VSAM   FILE          UC02
 TCLUC02    TRANCLASS     UC02
 UC02       TRANSACTION   UC02
 UC02       URIMAP        UC02
```

Figure 3-6 The logical components of another service instance using UC02 as identifier

In the evolution of Walmart to z/OS cloud services, services were initially defined in dedicated CICS servers on existing LPARs that supported traditional applications. Over time, this approach was changed to use dedicated parallel sysplexes. Walmart developers affectionately refer to these sysplexes as *CloudPlexes*, which are defined with multiple LPARS and CICS web servers.

Resources such as *max task* and *transaction classes* (and others) are used to control transaction volumes and to protect CICS servers from becoming overwhelmed during heavy bursts of requests.

Along with using common z/OS resource pools, like processor cycles and virtual storage, Walmart hosts services in clusters of multi-tenant regions. These service clusters serve as another form of a pooled resource. The address spaces themselves become a common asset shared by multiple tenants.

Walmart found through experience that they needed to complement the default basic control mechanisms with non-default configuration settings. For example, the following items were customized for CICS services:

- TCLASS: Controls the number of concurrently active transactions. An example of this control is provided in Figure 3-7 where you can see MAXACTIVE has been set to 100. MAXACTIVE is an attribute of the TCLASS, which is defined to CICS by using RDO/CEDA/DFHCSDUP. The TRANSACTIONCLASS (TCLASS) can be used to impose a control on the acceptable volume for a particular service instance. The key attributes of the TCLASS are MAXACTIVE and PURGETHRESH. These values represent the maximum concurrent number of active tasks by the ASID for this transaction (MAXACTIVE), and the number of requests that can be queued up before purging when MAXACTIVE is reached (PURGETHRESH).

```
OBJECT CHARACTERISTICS
 CEDA VIEW TRANCLASS( TCLUC01 )
  TRANCLASS     :TCLUC01
  GROUP         :UC01
  DESCRIPTION   :
 CLASS LIMITS
  MAXACTIVE     :100                    0-999
  PURGETHRESH   :0000005                NO | 1-1000000
 DEFINITION SIGNATURE
  DEFINETIME    : 01/15/16 13:09:37
  CHANGETIME    : 01/15/16 13:16:23
  CHANGEUSRID   : USER123
  CHANGEAGENT   : CSDAPI                CSDAPI | CSDBATCH
  CHANGEAGREL   : 0690
```

Figure 3-7 The TRANSACTIONCLASS controls volume of concurrently active transactions

- MAXPROCSYS: Controls the maximum number of processes per system. An example of this setting is found in Figure 3-8. The MACPROCSYS value is specified in the system parmlib member BPXPRMxx.

```
/****************************************************************************/
/*                                                                          */
/*    Specify the maximum number of processes that z/OS UNIX                */
/*     will allow to be active concurrently.                                */
/*                                                                          */
/*                                                                          */
/*     Notes:                                                               */
/*                                                                          */
/*          1. Minimum allowable value is 5.                                */
/*          2. Maximum allowable value is 32767.                            */
/*          3. If this parameter is not provided, the system default        */
/*             value for this parameter is 900.                             */
/*                                                                          */
/*                                                                          */
/*                                                                          */
/****************************************************************************/
MAXPROCSYS 3000                            /* System will allow at most 3000
                                              processes to be active
                                              concurrently              @P9C*/
```

Figure 3-8 The MAXPROCSYS value sets maximum number of processes

- MAXPROCUSER: Controls the maximum number of processes per user ID (or per ASID in this case). An example of this setting is found in Figure 3-9. The MACPROCUSER value is specified in the system parmlib member BPXPRMxx.

```
/****************************************************************************/
/*                                                                          */
/*    Specify the maximum number of processes that a single user            */
/*    (that is, with the same UID) is allowed to have concurrently          */
/*    active regardless of origin.                                          */
/*                                                                          */
/*     Notes:                                                               */
/*                                                                          */
/*          1. This parameter is the same as the Child_Max variable         */
/*             defined in POSIX 1003.1.                                     */
/*          2. Minimum allowable value is 3.                                */
/*          3. Maximum allowable value is 32767.                            */
/*          4. If this parameter is not provided, the system default        */
/*             value for this parameter is 25.                              */
/*                                                                          */
/*                                                                          */
/*                                                                          */
/****************************************************************************/
MAXPROCUSER(2500)                          /* Allow each user (same UID) to
                                              have at most 25 concurrent
                                              processes active            */
```

Figure 3-9 The MAXPROCUSER value sets maximum processes per UID

3.2 Taking advantage of z/OS system characteristics

Walmart initially used existing hosting environments for development activities. They already had clusters of regions dedicated to hosting SOA-style web services, so those environments were used for initial non-production deployment. Due to service consumption growth, they had to define new clusters for non-production activities.

However, for production environments Walmart took a more conservative approach. They cloned existing web service regions and defined new multi-region hosting environments for these new services.

Initial deployment involved new 12-region clusters per site to ensure an abundance of capacity for a new, potentially unpredictable type of workload and distribution of the regions across fault-zones (LPARs/CPCs) to ensure availability. These clusters were deployed into existing sysplexes and used existing capacity. After further adoption and related efficiency improvements, Walmart was able to justify dedicated sysplex environments and region clusters.

Having adopted SOA-style CICS web services before embarking on cloud service delivery, Walmart mostly used existing WLM constructs for their new environments. New ASIDs were included in their web services Service Class that carries a Very High Importance (1) and higher than average velocity (60) with NORMAL I/O priority as shown in Figure 3-10. These settings mirror the existing IBM OLTP-related service classes.

```
Base goal:
CPU Critical = NO      I/O Priority Group= NORMAL

    #  Duration   Imp  Goal description
    -  --------   -    ----------------------------------------------------
    1             1    Execution velocity of 60
```

Figure 3-10 Walmart used higher than average velocity

The ASID service class is used by the z/OS Workload Manager to ensure that performance goals (specified in business terms) are met. The activity associated with meeting that performance goal is the responsibility of the operating system. The actions that the system takes are related to the allocation of CPU and storage.

4

Elastic environment

Rapid elasticity *- Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.*

The NIST Definition of Cloud Computing
http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

This chapter discusses the attributes and capabilities of an elastic environment. It includes the following sections:

4.1 A first look at rapid elasticity

Paraphrasing the NIST definition, rapid elasticity is the capability to provision and release resources quickly and automatically in support of the ever-changing scaling requirements of your environment.

In a cloud environment, elasticity, or more precisely, rapid elasticity, is something that affects all aspects of a cloud-based approach to IT. Rapid elasticity is not a stand-alone feature or one that can be enabled. Instead, rapid elasticity is a fundamental characteristic that plays into all other areas of what makes a cloud implementation successful.

4.2 Elasticity planning considerations

Planning for elasticity is needed to facilitate adapting to workload demand. Planning not only prepares for the allocation of more resources during times of high demand, but also allows the scaling back of resources when they are no longer needed. This characteristic is closely related to the *pooled resource* characteristic, in that resource assignment and reassignment are necessary to achieve the elasticity.

Sometimes, either due to technological limitations or management approaches, elasticity is viewed as a consumer responsibility. The consumer must provision extra instances of a service to accommodate demand. This approach is typically more applicable to lower-level *IaaS* type resources.

Service providers can offer *auto-scale* capabilities that deliver automatic allocation of resources to meet increased demand combined with subsequent de-allocation of the resources as the demand subsides. Controls are needed to ensure that elasticity occurs within agreed-upon boundaries of service level agreements (SLAs) and operational level agreements (OLAs). This model aligns more closely to the style of hosting provided by IBM z/OS.

An interesting component of the *elasticity* characteristic is the scale-back/scale-down action and the primary driving factor behind it. This is a cost-controlling measure that is primarily concerned with avoiding over-provisioned resources. The unique nature of z/OS helps satisfy this characteristic due to negligible cost associated with certain resources being over provisioned.

From the consumer’s perspective, they have unlimited capacity available because (if the consumer accepts the cost) any amount of resources necessary to meet the consumer’s demand can be provisioned. However, this capacity is an illusion. Regardless of type, a service provider is still limited by the amount of physical capacity in their hosting environment, so planning remains important.

Parallel sysplex, although not required, does provide significant benefits associated with the elasticity characteristic. The ability to distribute requests both vertically within a system and horizontally across systems enables a robust degree of scalability.

4.3 Putting the plan into motion

With z/OS, the most straightforward approach to achieving elasticity is to over-provision the virtual runtime environments that host the services. You can then rely on the platform capabilities to ensure that those address spaces receive enough resources to satisfy demand. This might sound contradictory, given that a key objective of the elasticity characteristic in distributed models is to specifically avoid over provisioning. Or more precisely, the objective is to avoid the cost of over provisioning. The reason for this apparent contradiction lies in the ways that resource assignments and metering are handled across different platforms.

After a base z/OS hosting environment (infrastructure, LPARs, systems, subsystems) is in place, there is little to no cost for over-provisioning the higher-level virtual components necessary to provide a service. This in turn affects the level of abstraction where measuring and metering logically occur for a service. By using the granular resource assignments and the lack of exclusive resource allocation for idle assets that z/OS provides, cost assignments can be applied to actual usage of the resource rather than some concept of an active or running resource, avoiding the cost of over provisioning.

Another aspect of z/OS that enables this approach is the ability to scale resources independently. If a service function requires more processing capacity to satisfy a request, the system can assign more CPU to the service without also assigning units of memory and storage, providing another level of efficiency in resource usage.

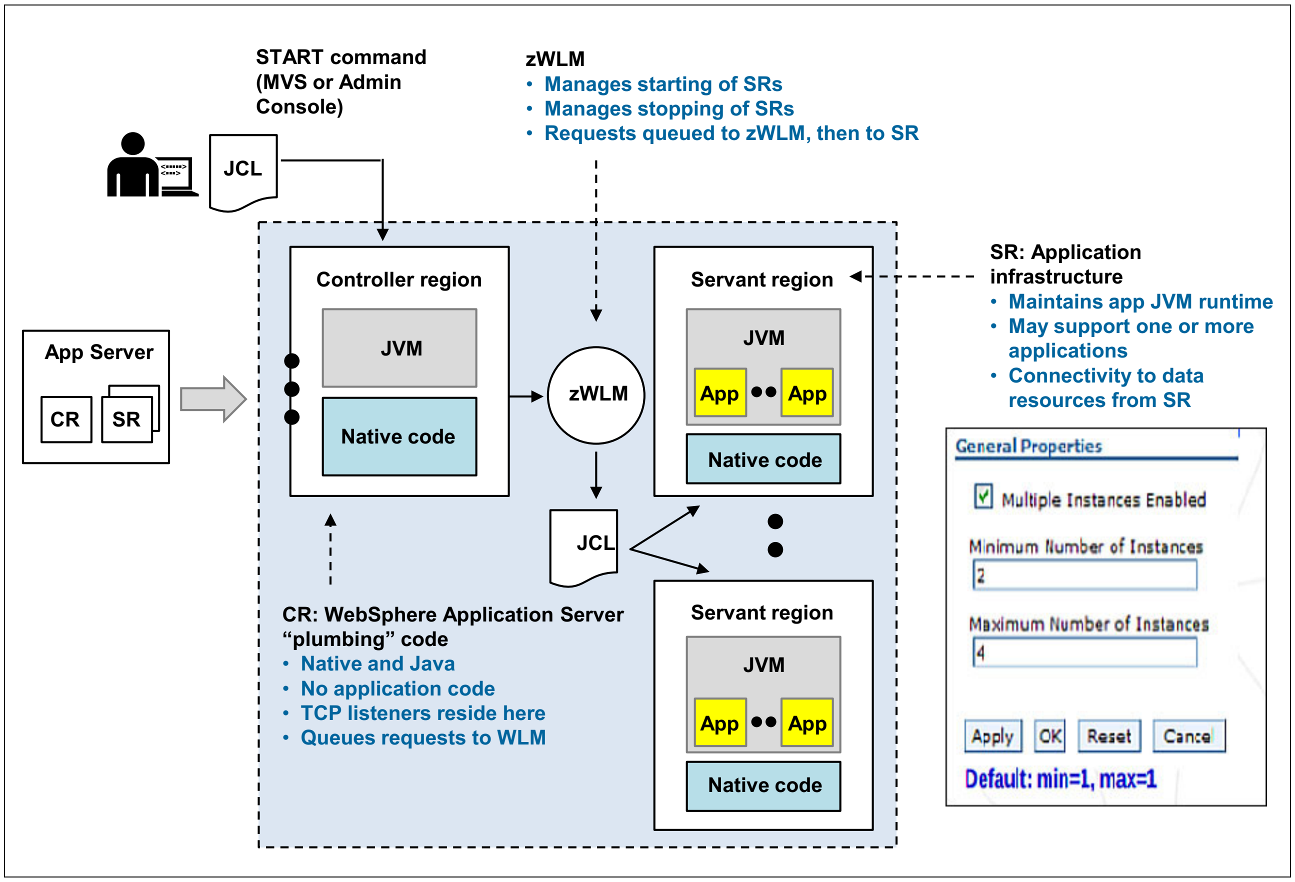
Parallel sysplex enhances this characteristic greatly. The ability to scale within a system and across systems provides great flexibility in resource distribution for elasticity, and allows for further platform mechanisms for efficiency. Enabling WLM-managed sysplex distribution ensures that the most eligible systems and address spaces are satisfying each request.

Parallel sysplex also provides a unique benefit called *data sharing*. The ability for numerous systems or virtual servers to process requests without the need to replicate and synchronize data among them can prove valuable. This attribute can be a key differentiator in identifying the types of services that are a good fit for the platform.

There might be situations where assigned resources are insufficient and extra resources need to be provisioned. This can take various forms, such as adding volumes to a storage pool or instantiating extra address spaces. Ideally, these actions can be automated and controlled by using a policy, and in some cases this capability already exists in system products or through third party tools.

Figure 4-1 provides a representation of Intelligent Dynamic Capacity Expansion (IDCE) feature of IBM WebSphere® Application Server[1].

In general, Figure 4-1 depicts the ability of an IBM WebSphere Application Server control region to use workload manager consultations. This enables the starting and stopping of servant regions based on demand. Although Walmart does not use this configuration, it is a good example of z/OS features related to elasticity.


START command (MVS or Admin Console)
JCL
zWLM
• Manages starting of SRs
• Manages stopping of SRs
• Requests queued to zWLM, then to SR
App Server
CR
SR
Controller region
JVM
Native code
zWLM
JCL
Servant region
JVM
App
App
Native code
Servant region
JVM
App
App
Native code
SR: Application infrastructure
• Maintains app JVM runtime
• May support one or more applications
• Connectivity to data resources from SR
CR: WebSphere Application Server "plumbing" code
• Native and Java
• No application code
• TCP listeners reside here
• Queues requests to WLM
General Properties
Apply
OK
Reset
Cancel

☑ Multiple Instances Enabled

Minimum Number of Instances: 2

Maximum Number of Instances: 4

Default: min=1, max=1

Figure 4-1 Graphical representation of the IDCE feature of IBM WebSphere Application Server

To effectively host an environment with this expansion characteristic, capacity planning remains as important as ever. This is true for any service provider, regardless of platform or public/private status. Physical infrastructure will be the limit of maximum capacity. Understanding macro-level usage trends and making data-driven projections is required to accommodate consumer demand.

Components like workload manager LPAR weighting, CPU VARY, and On/Off capacity on demand (CoD) can be used to ensure a dynamic environment. Except for LPAR weights at the PR/SM level, the Walmart implementation did not use these components. This indicates that the pieces that are needed for the cloud model are already in z/OS.

[1] Gary Picher's zWAS Elastic Scalability presentation

4.4 Service Examples

To over provision a hosting environment, define a group of address spaces that work as one such as high availability (HA) CICS regions. The group should consist of more address space identifiers (ASIDs) than those that would be defined for the anticipated workload (based on projected number of tenants, and previously defined controls like TCLASS). For example, you could initially define double the number of ASIDs than you expect to be necessary for the work volume. If you expected to need three ASIDs, you would define six as shown in Figure 4-2.

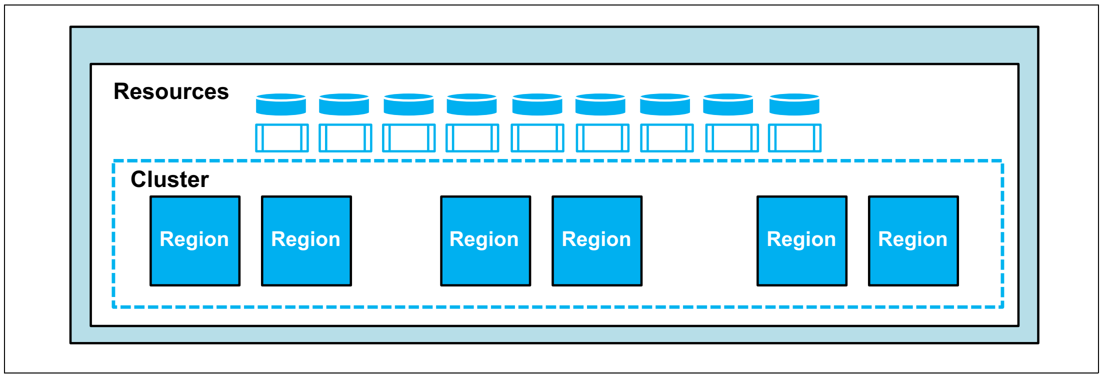


Figure 4-2 Over provisioning a hosting environment

As workload is driven against the service, resources are dynamically assigned and reassigned by the system to the appropriate ASIDs to accommodate the processing of the requests. Refer to Figure 4-3 on page 38.

By over provisioning the ASIDs, the system can absorb unexpected spikes or even partial outages without needing to modify the environment. This behavior demonstrates avoiding persistent resource allocations by the system. The resources can be assigned/reassigned where and how they are needed.

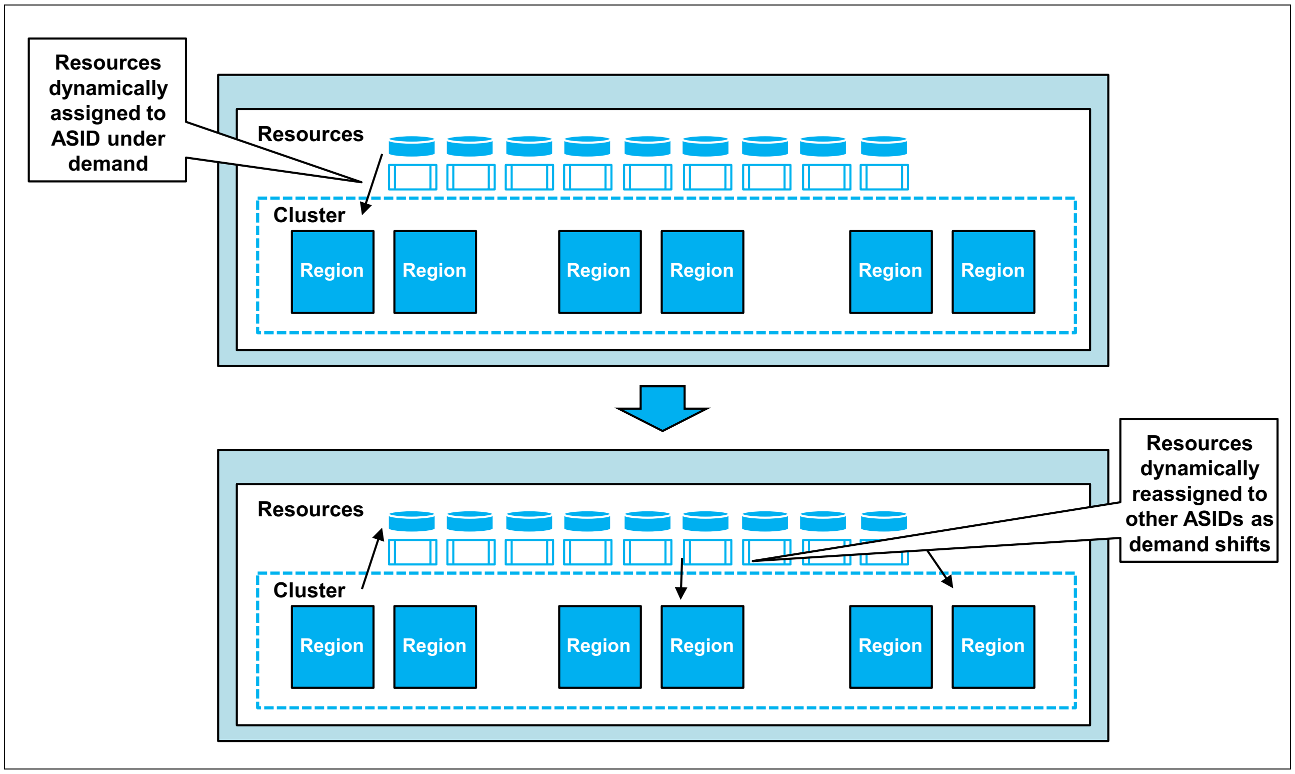

Resources dynamically assigned to ASID under demand
Resources
Cluster
Region
Region
Region
Region
Region
Region
Resources
Cluster
Region
Region
Region
Region
Region
Region
Resources dynamically reassigned to other ASIDs as demand shifts

Figure 4-3 Resources dynamically reassigned as demand shifts

An informed estimate is needed to establish an expected throughput per ASID. Estimates are typically done by benchmarking the resource usage of the service being provided and projecting the concurrent requests within the throughput controls established, and the ASID and system levels.

There will also need to be some initial resource capacity to satisfy the new workload. This can be accomplished by *borrowing* existing resources from other workloads during low activity times by using the same mechanisms that are used to provide resource pools and elasticity.

4.5 Experiences

When Walmart initially created region hosting clusters, they used some projections and assumptions to establish an arbitrary limit of 100 tenants per cluster. As adoption of the services increased, they soon exhausted this tenant limit and realized that they had been too conservative despite having plenty of resources for extra tenants. Walmart has since increased their initial imposed limits dramatically and are monitoring capacity to establish a cluster tenant target that is more reflective of actual usage.

Workload managed sysplex distribution with shared ports was used to take advantage of the strengths of the z/OS platform. A description of the full implementation of the TCP/IP configuration is beyond the scope of this book.

The environment must be configured to ensure that service requests are routed across the hosts effectively. This is accomplished through shared ports and, in an LPAR parallel sysplex environment, sysplex distribution.

To set up shared ports, the PORT statements in TCPPARMS are used to assign the same port number to each of the ASIDs and identify that port as shared. See Figure 4-4 for PORT statement examples.

```
PORT  12345   TCP   CICSRG1    NOAUTOL  DELAYA   SHAREP
PORT  12345   TCP   CICSRG2    NOAUTOL  DELAYA   SHAREP
PORT  12345   TCP   CICSRG3    NOAUTOL  DELAYA   SHAREP
PORT  12345   TCP   CICSRG4    NOAUTOL  DELAYA   SHAREP
PORT  12345   TCP   CICSRG5    NOAUTOL  DELAYA   SHAREP
PORT  12345   TCP   CICSRG6    NOAUTOL  DELAYA   SHAREP
```

Figure 4-4 PORT statements in TCPPARMS

The NOAUTOL, DELAYA and SHAREP parameters of the PORT statement are used to specify the following:

- NOAUTOL(OG): Prevent the ASID from being restarted after it has been stopped
- DELAYA(CKS): Explicit assignment of default setting that delays transmission of acknowledgements of packets received with the PUSH bit on in the TCP header
- SHAREP(ORT): Required to share port across multiple listeners

Note that a PORT statement such as **`PORT 12345 TCP CICSRG* NOAUTOL DELAYA SHAREP`** using generic region or server names will allow new regions to be provisioned and deployed without having to change the TCPPARMS PORT member.

Along with IPCONFIG SYSPLEXROUTING being defined in the TCPPARMS PROFILE, dynamic virtual IP addressing must be set up for the address and port that the ASIDs will be listening on as shown in Figure 4-5.

```
VIPADYNAMIC
VIPADIST DISTM SERVERWLM 123.45.67.89
   PORT 12345 XXXXX XXXXX XXXXX XXXXX
   DESTIP ALL
ENDVIPADYNAMIC
```

Figure 4-5 VIPADYNAMIC-VIPADISTRIBUTE statement in TCPPARMS

Figure 4-5 uses these statements:

- VIPADYNAMIC VIPADIST(RIBUTE) (DEFINE implicit): Enables the sysplex distributor function for DVIPA.
- DISTM(ETHOD) SERVERWLM: Specifies that server-specific values should be collected for this group of DVIPA ports. This allows Workload Manager (WLM) to assign weights to individual ASIDs for request distribution decisions.
- DESTIP ALL: Specifies that all TCP/IP stacks in the sysplex are targets for the DVIPA/Ports in this statement.

The combination of shared ports and WLM-managed sysplex distribution ensures requests are routed to the most appropriate system and ASID in the sysplex at that time. When applying this concept to the previously discussed resource pools (see Chapter 3, "Shared pool: Configurable resources" on page 25), the design can be represented as shown in Figure 4-6.

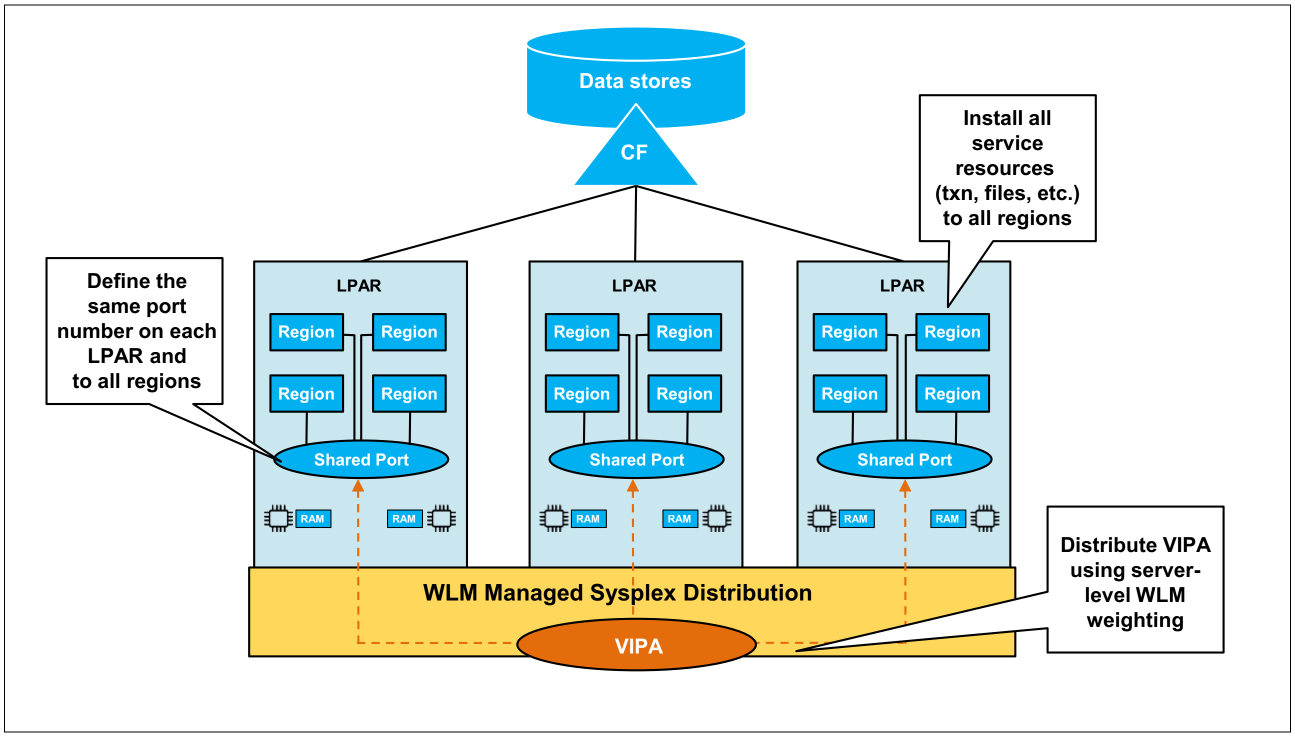

Data stores
CF
Install all service resources (txn, files, etc.) to all regions
Define the same port number on each LPAR and to all regions
LPAR
LPAR
LPAR
Region
Shared Port
RAM
WLM Managed Sysplex Distribution
VIPA
Distribute VIPA using server-level WLM weighting

Figure 4-6 WLM managed sysplex distribution

5

Ubiquitous access: ReST enabling your runtime

Broad network access. *Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms, such as mobile phones, tablets, notebooks, and workstations.*

The NIST Definition of Cloud Computing
http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

By using a ReST enabled runtime, access can be made ubiquitous, which means broadly available and convenient. Ubiquitous as an access attribute is consistent with the NIST definition of broad network access as an essential characteristic of cloud. Whether your users are on a desktop, tablet, phone, or other mobile device, you can enable ubiquitous access to your services by enabling your runtime with ReST.

The sections of this chapter provide an introduction to *broad network access*:

- 5.1, “An initial look at broad network access” on page 42
- 5.2, “An overview of ReST” on page 42
- 5.3, “ReST enabled runtime” on page 47
- 5.4, “How to deliver capabilities through web services” on page 47

5.1 An initial look at broad network access

The essential cloud characteristic of broad network access (ubiquitous access) basically dictates that capabilities are accessible through *standard interfaces* over a TCP/IP network. A key part of this description is the use of the term *standard*. Using *standards* for transport (as in TCP/IP and HTTP) and accessibility (as in URI and JSON) ensures that service capabilities are available to the broadest range of platforms and clients. A benefit provided by this model is the resultant abstraction and loose coupling of the clients from the specific systems of the service providers.

Broad network access is made possible through web services as shown in Figure 5-1. The process of delivering capabilities through web services is described in 5.4, "How to deliver capabilities through web services" on page 47.

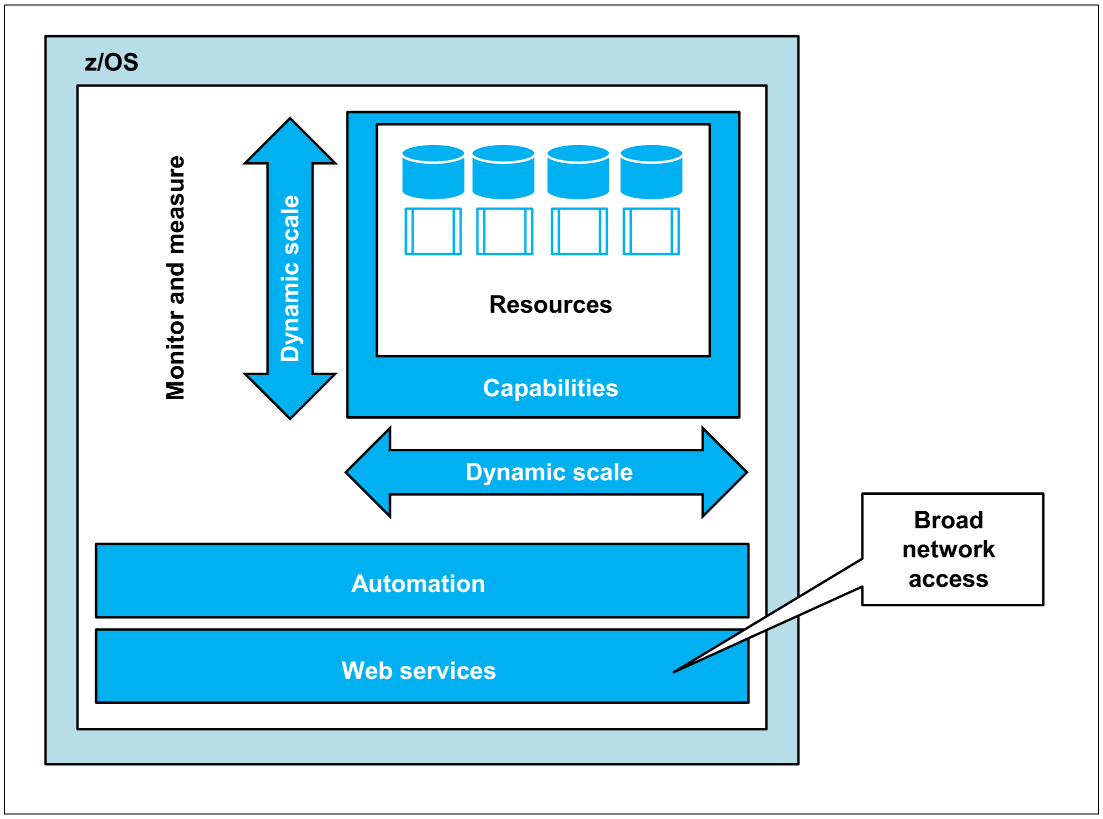


Figure 5-1 Web services provide broad network access

Although standards associated with SOAP web services were formerly the preferred option, ReST has become the de facto model in the cloud.

5.2 An overview of ReST

Almost anything you read about Representational State Transfer (ReST) will describe it as a design approach that uses a core set of HTTP operations to implement a set of web-based services. The design itself is referred to as *stateless*.

JSON is used as the data format in the examples in this book. However, JSON is not the only method of packaging and passing data. ReST services can handle JSON, XML, binary, documents, and plain text. ReST as a design approach is versatile in that different media or content types are supported, whereas SOAP is restricted to XML/WSDL only.

This publication narrows the description of ReST further to a view of ReST in an environment like that depicted in Figure 5-2. A ReST server accepts calls, invokes services based on those calls, and returns responses provided by the called services. Each of the services has access to data as needed to perform the requested service.

Each call contains (through HTTP protocol) everything that the requested service needs to accomplish a specific service as a self-contained operation. There is no need (or ability) for the service to recall or reference anything that is stored by a previous service call. The procedure can include these steps:

- The URL identifies the specific service.
- The HTTP request payload provides JSON for the service.
- The service can return (respond with) a JSON payload to the caller.
- The HTTP return codes (such as 200 for resource found and contents returned as JSON) can be used by the service to indicate results to the caller.

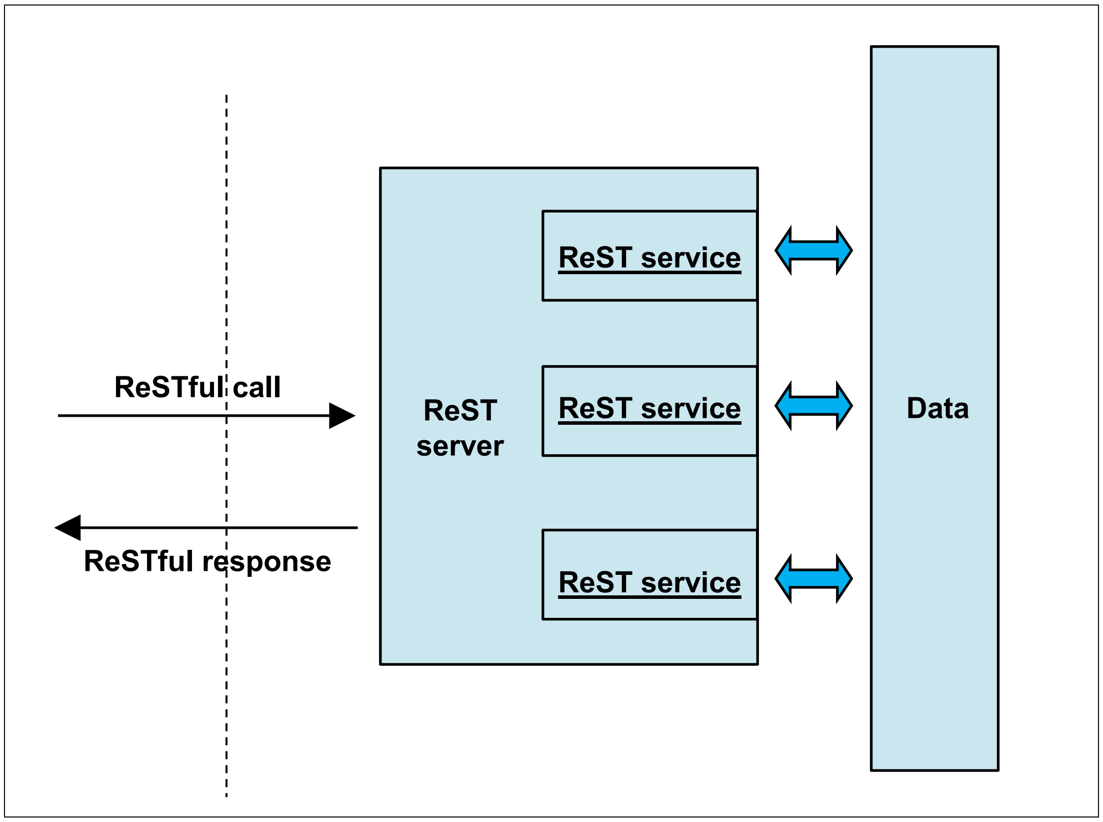


Figure 5-2 A ReST server and services

In this implementation, each call would be associated with only one of the set of operations that are often collectively referred to as create, retrieve, update, and delete. All data that is needed to complete the operation is made available as part of the call. No history or saved data from a previous operation is maintained by the service.

Each of the create, retrieve, update, and delete operations is associated with a corresponding form of HTTP as follows:

- Create is accomplished by using HTTP POST
- Retrieve is accomplished by using HTTP GET
- Update (change the state of a resource) is accomplished by using HTTP PUT
- Delete is accomplished by using HTTP DELETE

The URL associated with the HTTP call in some cases can contain what amounts to an exposed application directory. *Exposed application directory* means that the URL identifies the path as a directory structure that allows the ReST server to drill down to the service (application) that is to service the call.

This approach means that, as part of the design process, you will likely need to identify a set of primitives that corresponds to actions to be taken, and a data point or object to be acted upon. This set of primitives will be reflected in the structure of the HTTP calls and a corresponding set of well defined responses associated with create, retrieve, update, and delete operations.

5.2.1 Using z/OS Connect to ReST z/OS systems

The applications that you are probably most familiar with on z/OS systems are similar to the framework that enables solutions for mobile, social, and other user-facing apps. Transactions, data, and operations are always going to be at the heart of an enterprise solution. Although it is not a requirement for creating a cloud solution, IBM z/OS Connect has been designed to allow the application developer to easily ReST enable CICS and IMS-based programs, and even batch processes.

Having a unified solution in place through z/OS Connect also means less *one-off* implementations, which is helpful because *one-off* implementations can be troublesome to track and maintain. When implemented, a z/OS Connect solution streamlines the process of defining and then mapping these ReST calls to your existing z/OS operating system functions.

z/OS Connect is built on an IBM WebSphere Application Server Liberty profile for z/OS, and can be run as a *started task*. When configured, it acts as the cloud front end to your z/OS programs. Beyond providing that gateway, it also handles much of the complexity associated with parsing JSON data, System Authorization Facility (SAF) user authorization, and API publishing.

At the transaction level, z/OS Connect also allows for the creation of SMF records. SMF records are important for accountability in an enterprise environment, and for auditing and billing. These records allow administrators to view the amount of services consumed, when they are consumed or requested, the amount of data sent and received, and the overall response time. This technology is intended to allow the integration of mobile and distributed cloud consumers with the rich and robust landscape of enterprise IT, creating a truly *hybrid cloud* environment.

5.2.2 A brief overview of URI and URL

Many people use the term *URL* as a way to refer to something on the internet, where URL is short for Uniform Resource Locator. Even if you continue to use the term URL in a generic sense, be aware that a URL is a form of Uniform Resource Identifier (URI). The terms URL and URI are commonly used interchangeably. but there are distinctions between the two. A URL is a type of URI, but a URI can also refer to a Uniform Resource Name (URN).

The Internet Engineering Task Force (IETF) maintains a number of publications on the details of these concepts. You can also find many articles that describe URI, URL, and URN by using your favorite search engine. For the purposes of this document, URL is the only relevant type of URI described. Further discussion of this relationship is beyond the scope of this publication.

The term URL is also covered in 5.4.2, "The design of URIs for the ReSTful services" on page 47. More description of URLs can be found in 5.4.4, "Experiences" on page 50, where lessons learned by Walmart in the context of an API and URL are provided.

5.2.3 Using a ReST API: An example

This section provides an example of a ReST API using an application called XYZ that provides four services. As you might expect, those services are create, retrieve, update, and delete. Table 5-1 summarizes the four types of HTTP calls that would most likely be defined in the documentation for this example application. The table shows four types of calls (POST, GET, PUT, and DELETE) to `www.xyzdomainname.com/xyz/XYZthing/`. Your API would involve more detail, but this simplified example shows what needs to be considered when designing and implementing a ReST API.

Table 5-1 Summary of example ReST API calls

URL	What is being requested	Passed	Returned[a]
HTTP POST with www.xyzdomainname.com/xyz/XYZthing/	Create an XYZthing	JSON with data that should be placed in the newly created XYZthing	Code=201 indicates that the request has been completed successfully and a new XYZthing has been created. With Code = 201, an XYZthingKey is returned in JSON. The XYZthingKey must be specified on subsequent calls to access or manipulate this XYZthing.
HTTP GET with www.xyzdomainname.com/xyz/XYZthing/<XYZthingKey>	Retrieve the XYZthing associated with the key <XYZthingKey>		Code=200 indicates that the requested XYZthing has been found and its contents returned as JSON
HTTP PUT with www.xyzdomainname.com/xyz/XYZthing/<XYZthingKey>	Update the XYZthing associated with the key <XYZthingKey>	JSON that defines the data to be updated	Code=200 indicates that the requested XYZthing has been found and its contents have been updated per the JSON that was provided.
HTTP DELETE with www.xyzdomainname.com/xyz/XYZthing/<XYZthingKey>	Delete the XYZthing associated with the key <XYZthingKey>		Code=200 indicates that the requested XYZthing has been found and deleted

a. For a complete list of return codes (HTTP status code assignments) and their meaning, see:
`http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml#http-status-codes-1`

The URL column in Table 5-1 on page 45 contains cells that describe a URL and the type of call. The next column in each row describes the service *that is being requested* in the context of create, retrieve, update, and delete. The third column (*Passed*) describes what the caller would provide in the call. Finally, the returned column indicates the code that is returned (such as 201) to indicate success or failure.

Consider the scenario of requesting the creation of some thing and then making a request to retrieve the contents of what was previously created. In Table 5-1, note that POST and GET are the two operations or types of HTTP calls that accomplish this process.

Figure 5-3 demonstrates the flow that is associated with an application calling a service to create some form of data object that can be retrieved later. The operations and responses occur in sequence from top to bottom. The ReST Services Server directs the call to the service (*xyz* in this example). The xyz service creates the data or “thing” and returns a corresponding key or some form of identifier back to the caller as part of the returned HTTP JSON payload.

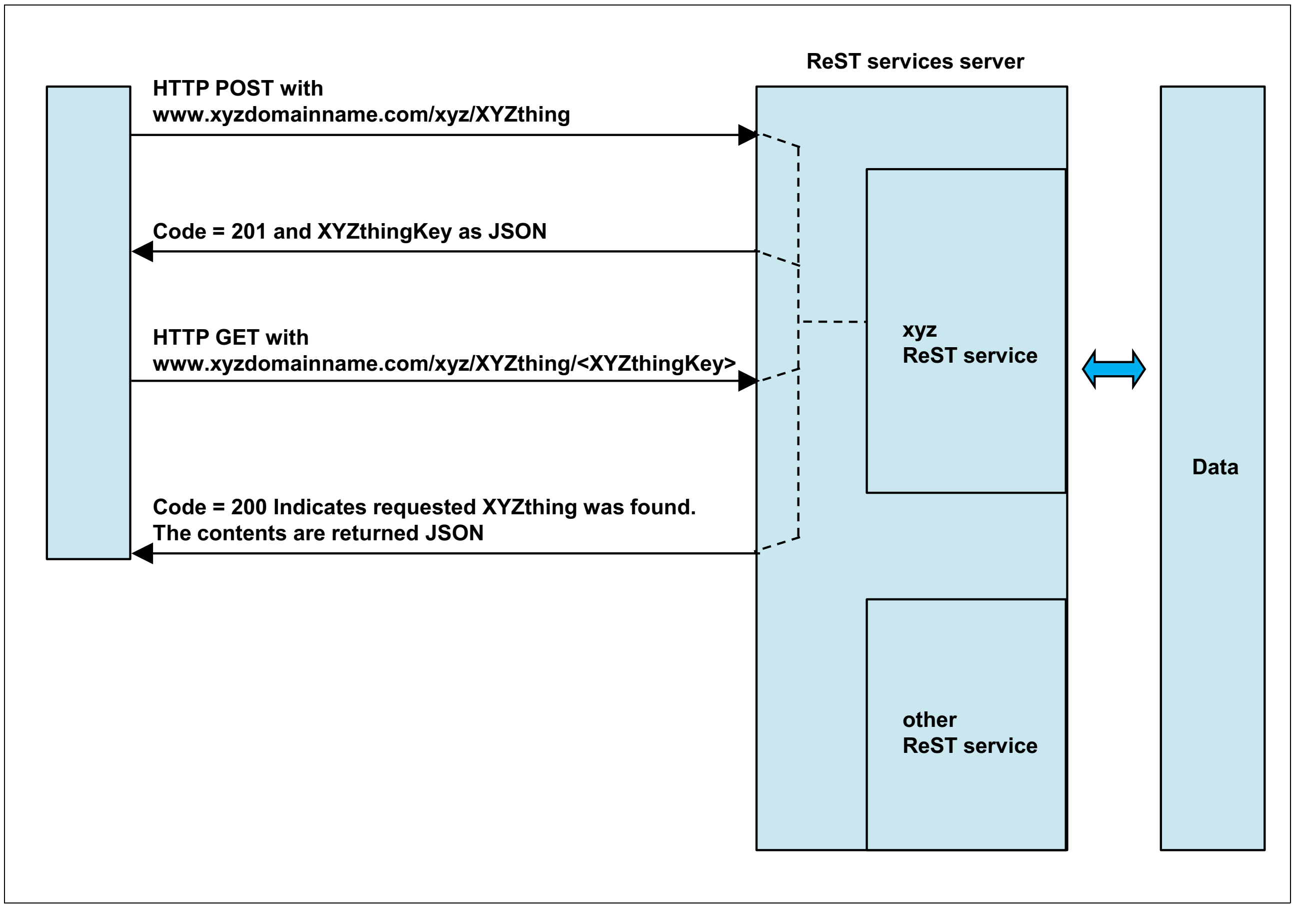

ReST services server
HTTP POST with www.xyzdomainname.com/xyz/XYZthing
Code = 201 and XYZthingKey as JSON
HTTP GET with www.xyzdomainname.com/xyz/XYZthing/<XYZthingKey>
Code = 200 Indicates requested XYZthing was found. The contents are returned JSON
xyz ReST service
other ReST service
Data

Figure 5-3 Sequence of ReST API calls

When the application (on the left) needs to retrieve the data that was previously created, it makes a call to the service that provided the *key* that was returned at the time of the creation. On this second call (in the form of HTTP GET), the ReST services server directs the request to the XYZ service with the *key* identifier. The service retrieves the data associated with the key that was provided on the call, and returns the data back to the caller as part of the returned HTTP JSON payload. The caller of the service (the application on the left) is able to access the payload when it sees the HTTP return code of 200. These codes are defined as part of the HTTP protocol.

5.3 ReST enabled runtime

You might think that because in a ReST enabled runtime your services have no knowledge, memory, or recollection of any previous communication with a client (*the caller or consumer of a service*), that ReST is bad to use. However, this environment is good in the sense that it does not matter what instance of a service receives and acts on a request. If it does not matter which instance of a service provides the requested service, you can implement multiple servers. In addition, the actual number of servers can be provisioned as needed and then deprovision when not needed. Your ReST enabled runtime can therefore exist in an elastic environment. For this reason, ReST is a good choice for a cloud implementation.

5.4 How to deliver capabilities through web services

Delivering capabilities through web services involves both a journey and a goal. The journey consists of incremental steps. With each step, you gain knowledge and adjust your plans for future steps. What you learn can cause you to revisit previous decisions and modify your approach to implementing a service.

To deliver capabilities through web accessible services, you must establish an access layer with a collection of some type of web servers. Generally, use a related cluster of address spaces (resource pool) providing hosting for multiple tenants.

Web service capabilities are already provided by many existing facilities such as IBM CICS Transaction Server for z/OS, IBM WebSphere Application Server Liberty Profile, and IBM Information Management System (IMS) Transaction and Database Servers (IMS/TS). Beyond the web services themselves, you need to determine which back-end capabilities you want or need to expose using web services and how to tie them together. This can be accomplished with many native mechanisms. You can also use the z/OS Connect software to assist with this planning.

5.4.1 Managing network resources

Service providers need to establish how they will manage their network resources. This decision will likely depend on the type of runtime environment in use. Some environments are conducive to full abstraction and require less management.

5.4.2 The design of URIs for the ReSTful services

Great care is needed in the design of URIs for ReSTful services. However, the details of ReSTful API design are beyond the scope of this document. Fortunately, many online articles and documents are available that address ReSTful API design. The following are general suggestions for the design process:

- Design your API so that it is easy for developers to use.
- Use nouns rather than actions (or verbs) in your URLs.

  For example, by using `/persons` on a URL in a context such as `www.asample.com/persons`, an HTTP Post, Get, Put, and Delete can determine or trigger the desired action relating to persons and the resource or resources associated with those persons.
- Review 5.4.4, “Experiences”, which describes Walmart’s experiences and lessons learned relative to the API and related URLs.

ReSTful API design is important when assigning individualized URIs for each tenant to support isolation and assist with measuring and metering. For more information, see Chapter 6, “Metering and measuring” on page 53.

One of the components of an instance of a service is the relationship to the URI. In the CICS examples used in earlier chapters, this component is the URIMAP. Figure 5-4 shows the partial output `CEDA View Urimap( UC01 )`.

```
CEDA View Urimap( UC01 )
Urimap      : UC01
Group       : UC01
DEScription :
STatus      : Enabled Enabled | Disabled
USAge       : Server  Server | Client | Pipeline | Atom
                      | Jvmserver
UNIVERSAL RESOURCE IDENTIFIER
SCheme      : HTTP    HTTP | HTTPS
POrt        : No      No | 1-65535
HOST        : *
(Mixed Case):
PAth        : /root/type/org/tenant/resource*
(Mixed Case):
```

Figure 5-4 View of assignment of the URI

Note that the UC01 specified in the CEDA command corresponds to the embedded identifier in the name that associates it to the other logical resources. The logical names of other resources can be seen in Figure 5-5.

```
EX G(UC01)
ENTER COMMANDS
 NAME      TYPE         GROUP
 UC01VSAM  FILE         UC01
 TCLUC01   TRANCLASS    UC01
 UC01      TRANSACTION  UC01
 UC01      URIMAP       UC01
```

Figure 5-5 Identifiers associated with logical resources of the service instance

As with the internet at large, IP addresses are not commonly used by consumers to access these network locations. Names are assigned to the IP addresses through a domain name server (DNS) to provide easier consumption. The same expectation exists with this implementation.

As shown in Figure 5-6, a DNS A record (Address record, which is not to be confused with an Alias Record) should be assigned to the VIPA to provide meaning and abstraction. Defining a DNS ALIAS or CNAME record enables even further abstraction and provides greater flexibility in the management of the service with little to no impact to the consumers. The z/OS administrators generally do not have responsibility for DNS management, so collaboration with other teams is often necessary for this design component.

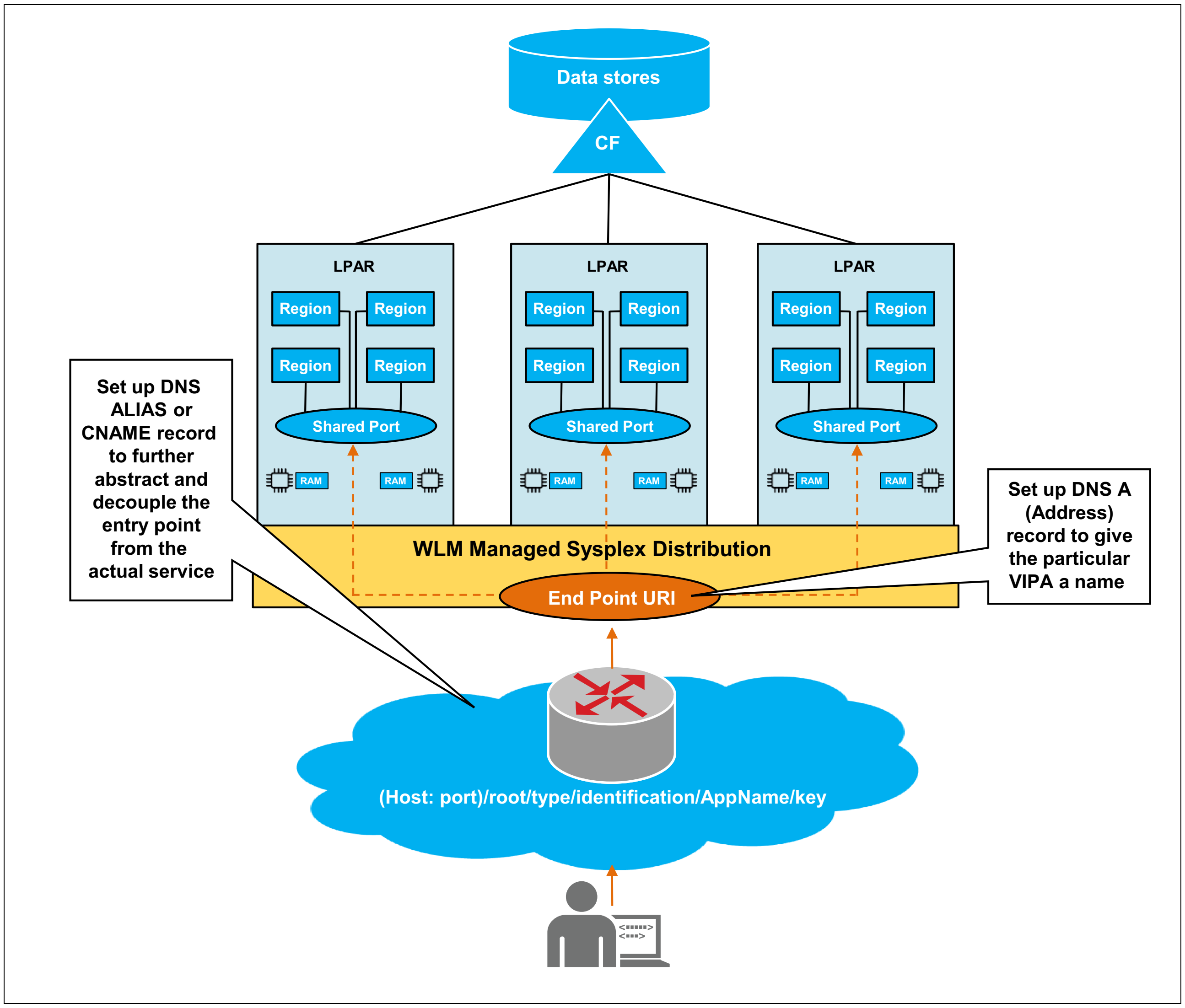

Data stores
CF
LPAR
Region
Shared Port
RAM
LPAR
Region
Shared Port
RAM
LPAR
Region
Shared Port
RAM
Set up DNS ALIAS or CNAME record to further abstract and decouple the entry point from the actual service
WLM Managed Sysplex Distribution
End Point URI
Set up DNS A (Address) record to give the particular VIPA a name
(Host: port)/root/type/identification/AppName/key

Figure 5-6 DNS ALIAS provides abstraction

From the architectural view of the environment, adding meaningful name constructs (DNS names and URIs) to the design helps tie the pooling and elasticity constructs together into a more easily understood view.

5.4.3 Security considerations

Security is also a critical consideration. The scope of the network to which z/OS based services are made available has a significant bearing on how security, *particularly authentication*, is implemented.

Basic authentication can be sufficient if all clients are effectively surrogates for other clients and the surrogate is protected by a DMZ. However, extra security layers should be considered to safeguard services from all attack vectors.

Until additional capabilities are included in the platform, the most straightforward approach to handling authorization controls is to define surrogate user IDs for authorization to underlying service resources. This approach also helps track resource usage for the measured service characteristics. Encryption should also be used in most cases as well as maintaining support for current Transport Layer Security (TLS).

The role of AUTHIDs should also be considered as you plan for cloud services and support for multiple tenants. Walmart initially used a set of predefined groups of IBM RACF® IDs. These IDs were used as needed to support multiple tenants.

As Walmart's cloud services implementation evolved, a new service was created that eliminated the need for the pool of RACF IDs. The new service provided for the dynamic creation of RACF IDs.

The DNS assignments, particularly the ALIAS, can play a major role in the design of security around the service, as well. For information about implementing this and other security considerations when creating cloud services that are hosted in CICS, see *How Walmart Became a Cloud Services Provider with CICS*, SG24-8347.

5.4.4 Experiences

The first service implemented in this model by Walmart had specific requirements for the API. The service was developed to replace an off-the-shelf product that had already been coded and deployed. Therefore, Walmart developers did not have the option to design the API. Instead, they had to develop their new service so that it adhered to the existing API. The new service was required to imitate the API already in use by the product it was replacing.

This API structure worked for the initial use case. However, that existing API structure was not ideal as a pattern for a more general, repeatable approach. A new URI pattern was designed following the principles of the cloud delivery model in that uniqueness and convention were employed. However, even with a significant amount of forethought, some lessons were learned by the Walmart development team and further adjustments are being considered.

Examples of lessons learned and adjustments being considered include modifications to consumer identity and versioning. Some of the lessons learned are provided here for your benefit.

One example of an adjustment under review centers on the components of a URI that are used to uniquely identify tenants. The initial approach was to embed identifiers as nodes within each assigned URI that established organizational accounts and tenant identities by name:

```
(Host:port)/root/type/org/tenant/resources
```

Although `(Host:port)/root/type/org/tenant/resources` is definitely a valid approach, other approaches warrant review for applicability and consistency in a broader API environment. Some examples of other common approaches are listed here:

- Maintain tenant identity in the URI, but obfuscate the identity by using an access key (typically with a UUID or similar value):

  ```
  (Host:port)/root/type/{AccessKey}/resources
  ```

- Use an access key passed on a query string:

  ```
  (Host:port)/root/type/resources/key?TenantId={AccessKey}
  ```

- Use tenant identity as a subdomain on the DNS assignment:

  ```
  TenantID.example.com/root/type/resources
  ```

- Remove tenant identity from the URI completely, and rely on headers, authentication parameters, or tokens:

  ```
  (Host:port)/root/type/resources/key
  (Determine tenant from Authentication: Basic HTTP header)
  ```

At the time of writing, Walmart has not made a change to their existing approach. Options are currently being reviewed with the intention of establishing a common approach to multi-tenant APIs across the enterprise.

Additional approaches that take versioning into consideration

As this book was being written, Walmart was planning a change in API design that involved including versioning in the URI. Most changes to the underlying functions provided by a service are intended to be incremental, and they have been so far. However, more significant releases might include changes (so-called *breaking-changes*) that materially affect the functionality of the service and might impact or even break existing applications that are using the service. In these cases, a versioned API provides an easy way for the consumers to accept or reject a change.

Semantic versioning, which includes version, release, and patch identification, is followed in the internal management of the capabilities, and should be reflected at some degree in the API. At a minimum, the version and release levels will be included in new URIs going forward as shown in this example:

```
(Host:port)/root/type/version/org/tenant/resources
```

5.4.5 Ubiquitous as an essential attribute

By using a ReST enabled runtime, access can be made ubiquitous, which means broadly available and convenient. Ubiquitous as an access attribute is consistent with the NIST definition of broad network access as an essential characteristic of cloud. Whether your users are on a desktop, tablet, phone, or other mobile device, you can enable ubiquitous access to your services by enabling your runtime with ReST.

6

Metering and measuring

Measured service. *Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported providing transparency for both the provider and consumer of the utilized service.*

The NIST Definition of Cloud Computing
`http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`

The automated collection of resource usage metrics is essential. These metrics are key in determining the impact of the dynamics of demand on resources and the related ability to meet service level goals and agreements. In addition, usage metrics are needed to support the billing process.

This chapter provides an overview of the concepts of metering and measuring in a cloud services environment in the following sections:

- 6.1, "An initial look at metering and measuring" on page 54
- 6.2, "How-to considerations" on page 55
- 6.3, "Service example" on page 55
- 6.4, "The criticality of z/OS metering and measuring" on page 59

6.1 An initial look at metering and measuring

Metering and measuring is the process of capturing data about resource usage. The ability to measure and record usage of resources is an inherent function of z/OS as shown in Figure 6-1.

Consumers typically receive an abstracted, higher-level contextual view of the usage data. This information is commonly provided on a per-unit basis such as the number of requests, number of users, MB used, and so on. This abstracted level of per-unit reporting facilitates the calculation of the aggregate costs of lower-level resources. This calculation of per-unit usage in turn enables a billing process to recover the cost of providing services.

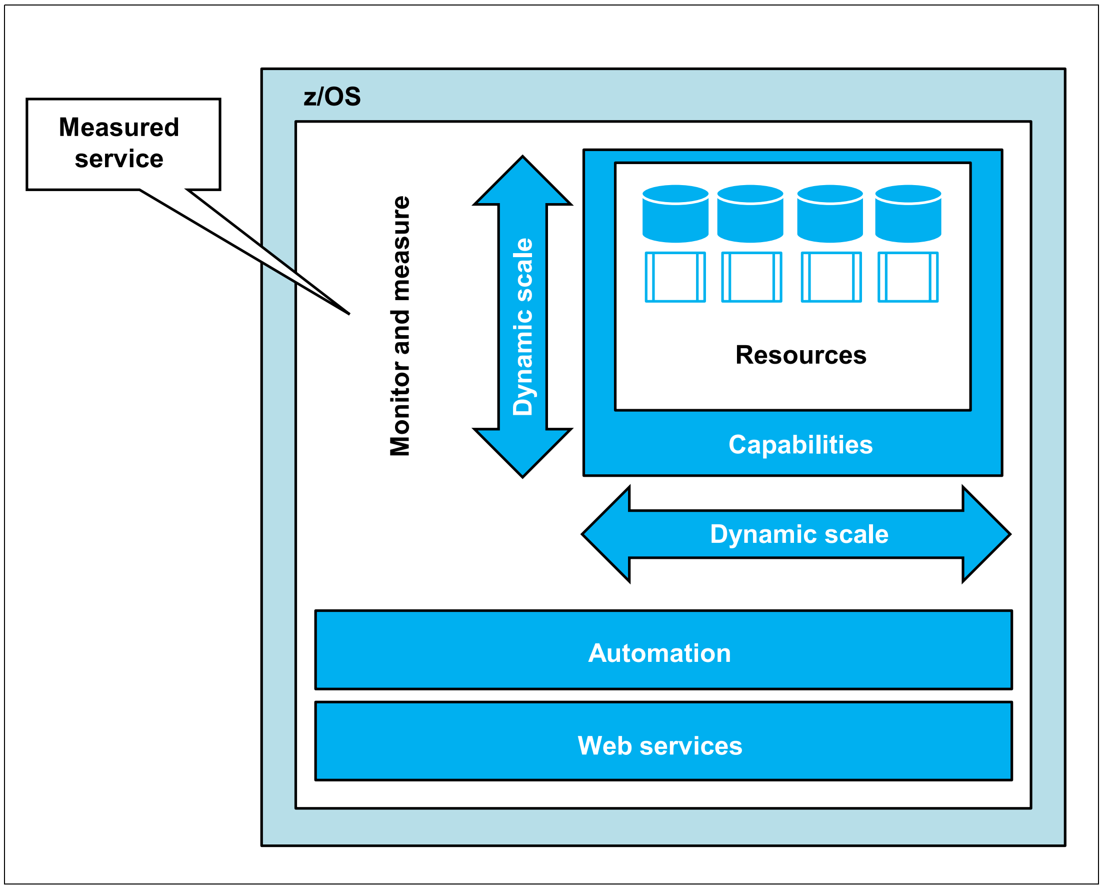


Figure 6-1 Measured service

In addition to recovering costs, the abstracted view or calculations can be useful for service providers to examine operational health. From an operational perspective, resource assignment decisions and other operational decisions can be based on collected data. The decisions that are made might require manual intervention or could occur programmatically to dynamically change resource assignments, availability, and scalability.

A benefit of this visibility, particularly from the consumer perspective, is that it helps promote self-governance of resource usage. Consumers can monitor the cost of their usage on an on-going basis. The consumers then have the data that they need to make informed decisions on continued operations, usage, and service level requirements or expectations.

6.2 How-to considerations

Well designed naming conventions can be useful in identifying and reporting on resources. Having identifying characteristics within the names of resources such as data set names and transaction IDs can make it easier to tie resources together, and to a consumer with reporting tools. However, naming conventions must be balanced with the naming restrictions associated with each resource type. Naming conventions might not be possible in all cases. When naming conventions are not appropriate or implementable, then other mechanisms must be used to track usage.

Regardless of the mechanisms that are used, resource usage must be identifiable and assignable to individual tenants.

Cost models should be developed that allow roll-up to an applicable per-unit metric. For a less complex service, cost models can simply include the cost of disk space or network bandwidth, or a combination of both. A more complex service can include a broader array of resources that need to be accounted for. The allocated portions of those various resources should be aggregated up to a simpler metric. For example, a service might require accounting for CPU usage from multiple address spaces as well as disk usage, I/O, and network bandwidth.

The IBM System Management Facilities (SMF) data contains most of the information that is needed to track resource usage at the appropriate level, but it is not readily consumable by default. The data (relevant record types/sub-types) typically needs to be extracted and then formatted before it can be queried in a meaningful way. Although the data is available, service providers need to include off-the-shelf tools or build processes for queries and reporting to display the information in a useful format.

The currency of the data being reported depends on SLAs that are established with the provider's consumers. These SLAs influence how data is collected, used, and reported.

This data must be accessible to the service consumers to provide visibility into their usage. When made available, this data provides information that helps consumers make fact-based decisions on application investments

6.3 Service example

This section provides an overview of a service in the context of capturing usage data, a cost model, and reporting.

6.3.1 Capturing usage data

Understanding which resources comprise a service is the first step in enabling data capture. The appropriate SMF record type that is required to track the relevant resources must be enabled in the system. The record types are enabled by using the SYS(TYPE()) statement in PARMLIB member SMFPRMxx. The example service group, UC01, is shown in Figure 6-2 so you can see how the basic logical resources of UC01 are defined.

```
EX G(UC01)
ENTER COMMANDS
 NAME      TYPE          GROUP
 UC01VSAM  FILE          UC01
 TCLUC01   TRANCLASS     UC01
 UC01      TRANSACTION   UC01
 UC01      URIMAP        UC01
```

Figure 6-2 UC01 service group

Some measurable resources for the service include disk storage and I/O bandwidth for the FILE, processor and memory for the TRANSACTION, and network bandwidth associated with the URIMAP. With these resources in mind, appropriate SMF record types can be identified.

These record types might be considered for measuring this service:

- TRANSACTION:
  - Type 30 and Type 7x (address space and processor)
  - Type 110 (CICS)
- URIMAP: Type 119 (TCP/IP)
- FILE: Type 42 (SMS and I/O)

SMF might not be the only source of data for measuring usage. A good example of this situation is related to disk storage. For a simple view of disk space usage, it might be more effective to use native system utilities such as DFSMS Data Collection Facility (DCOLLECT) to quickly acquire this information and associate it with other data sources.

Although not required, enabling SMF recording to log streams improves the accessibility of the data. With log stream recording enabled, the most recent data can be easily extracted at any time with the IFASMFDL utility.

With the system configured for efficient access to the relevant measurement data, you can use the naming conventions to facilitate capturing the data at the required level of granularity and association. The architectural view of a sample environment is shown in Figure 6-3 that includes these considerations for capturing measurement data. The design represents a system of resource pools that supports elasticity in delivery of services, is broadly accessible through open network-related standards, and maintains measurement of service resource usage.

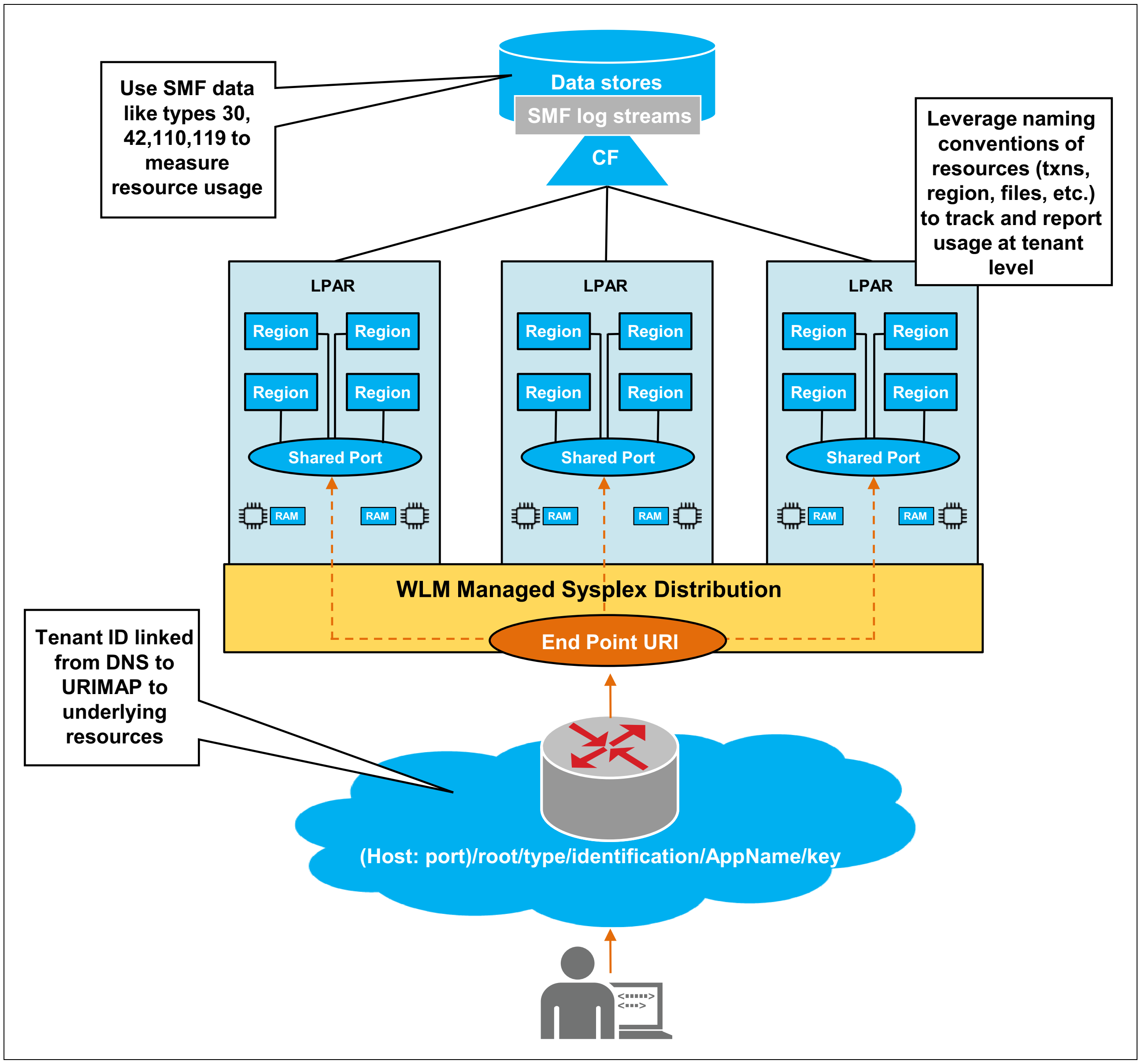

Use SMF data like types 30, 42,110,119 to measure resource usage
Data stores
SMF log streams
CF
Leverage naming conventions of resources (txns, region, files, etc.) to track and report usage at tenant level
LPAR
Region
Shared Port
RAM
WLM Managed Sysplex Distribution
End Point URI
Tenant ID linked from DNS to URIMAP to underlying resources
(Host: port)/root/type/identification/AppName/key

Figure 6-3 Capturing usage data

6.3.2 Reporting data

Determine what types of views of the data are needed based on the target audience and expected use for the reporting. As previously noted, service consumers typically need a higher-level abstracted view of activity, and service providers benefit from the lower-level details for operational considerations. One approach is to provide a first-pass detailed view to providers, and a second-pass summary view to consumers:

- For transactions and processor, the service provider might be interested in actual CPU time per request, time spent on different TCBs, and dispatching time, whereas consumers are just interested in total transaction count and average response time.
- For disk and I/O, the service provider might be interested in extent distribution, cache-hit ratios, and individual service time components, whereas consumers are interested in total disk space usage and average response time.

Presentation of the information to the consumer is also important. Some of the same principles of the cloud model should be followed in the delivery of the reporting. Consumers should receive an individualized view of their, and only their, usage information. The reporting should be provided automatically, typically along with their service instance. Access to the reporting should be broadly accessible over the network, so HTML-based presentation is beneficial. If the data formatting adheres to open standards like JSON, many open source libraries exist that can greatly facilitate this type of presentation.

6.3.3 Cost model

Although not explicitly stated in the measured service description in 6.3.1, “Capturing usage data” on page 56, reporting the cost of the usage of a service is an important use case associated with this characteristic. The ability to assign resource costs to a service instance is therefore key.

The resource costs allocated to a service will inevitably vary dramatically among different types of services and across different providers. Due to this variability, no clearly defined guidelines exist for determining cost models. The costs of hardware assets like processors, memory, and storage are relatively easy to identify and commonly included in a service cost. The costs of other resources such as environmental (power, floor space, and cooling) or licensing charges might be more difficult to allocate, or might already be rolled into other resource costs.

Another consideration in defining a service cost is the distribution of cost over the expected life span of the asset. This distribution can be accomplished by spreading the aggregate cost of a resource pool over the depreciation schedule for that asset, then dividing that yearly cost across days or months and across consumable units (for example, per-MB for storage, or per-MIPS for processor). All of these considerations, among others, need to be taken into account when building the cost model for a service.

6.3.4 Experiences

Although the currency of measurement data is ultimately up to the agreement between the consumer and provider, Walmart has determined that the closer to real time this information can be provided, the better. Walmart converted SMF recording to log streams years ago, so the data was already readily accessible, but still relied on third party tools and manual reporting processes for presentation to the service consumers.

In an environment-wide effort to increase the currency of measurement data, Walmart deployed an exit program into the CICS XMNOUT Global User Exit Point to capture transaction resource information as it is being passed to SMF. The data is immediately formatted, associated with other SMF record types being automatically pulled from the log streams, and loaded into an IBM DB2® database so that it can be queried.

This process was a joint effort between the engineers in the (Walmart) Platform Services Development/Delivery group and the (Walmart) Capacity Planning & Management group because the results were mutually beneficial to both areas.

To provide individualized views of this information to their consumers, Walmart created a generic HTML5/JavaScript page that is delivered as part of each service instance (an extra URIMAP resource included in the Service Group). The resource usage information in the database is continually queried, formatted as a JSON array, and stored in a ReSTful cache service by a background transaction.

When a consumer accesses the resource measurement page, the JavaScript continually polls the cache service and refreshes the page with graphs of service activity history along with average response times and current availability status. This display gives consumers direct access to an isolated view of metrics, while giving Walmart (as providers of the service) flexibility in presentation and making adjustments to meet the needs of the customers.

6.4 The criticality of z/OS metering and measuring

Detailed metering allows a cloud service provider to understand the total cost of running a particular service. As the market becomes more competitive for services, the need for detailed metering becomes more critical. The metering capabilities inherent in z/OS enable a service provider to fine-tune usage rates based on actual consumption.

As an installation moves to a usage-based model, it becomes easier for z/OS-based services to demonstrate their financial value. This paradigm of usage accounting transparency meets the demands of an increasingly savvy cloud user community.

7

Self-service provisioning

On-demand self-service. *A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.*

The NIST Definition of Cloud Computing
`http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`

This chapter provides an introduction to the following topics in the context of Walmart's experiences:

7.1 Self service overview

After you establish an overall design that enables the other cloud characteristics, you must provide access to the services in an on-demand, self-service manner. To provide self-service access, you need to automate the end-to-end creation of a service instance and provide easy access to this capability to the consumers. The basic steps to accomplish access to services in a self-service environment include identification of service components, programming or scripting these components, and exposing a method for triggering the automated process.

Figure 7-1 provides a view of on-demand self-service in relation to other aspects of z/OS and the cloud.

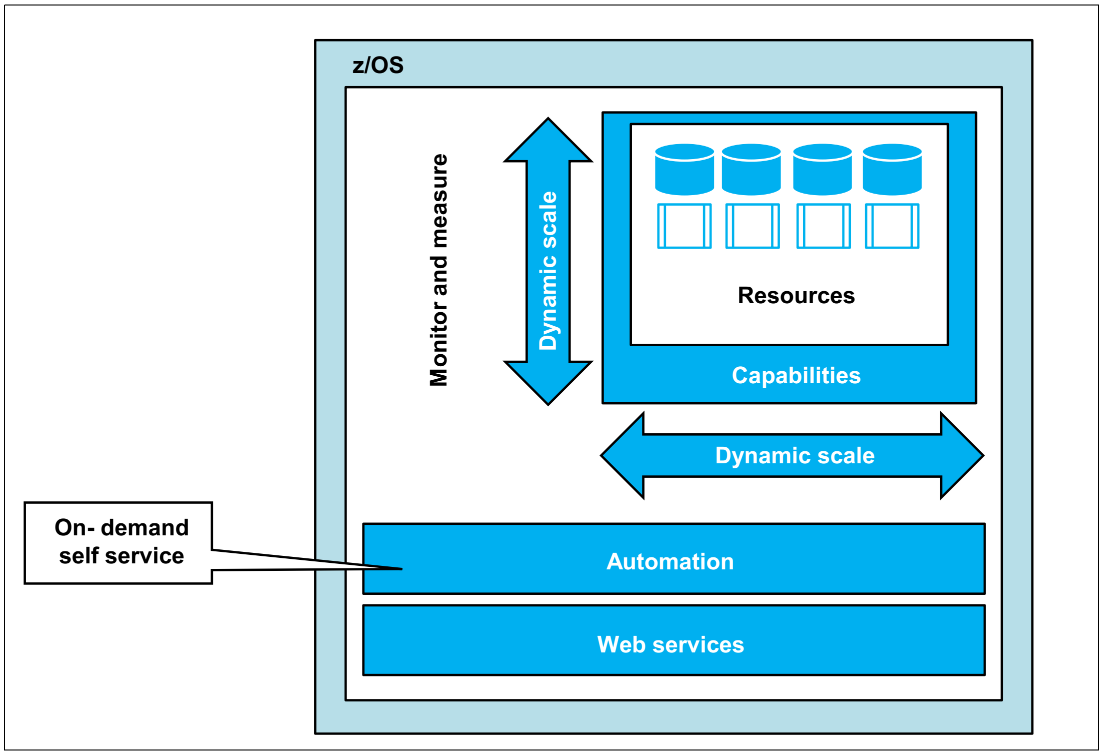


Figure 7-1 On-demand self service

Although not explicitly shown, provisioning of the service is expected to be automated. Even with the absence of consumer-to-provider interaction, any manual provider actions on the server should be avoided.

7.2 How to implement self-service provisioning

There is no clearly defined formula for how to accomplish self-service provisioning. Implementations vary greatly from environment to environment, making it impossible to prescribe a specific approach to achieve this cloud characteristic. However, a reasonable general approach consists of three basic tasks: Identifying, programming, and exposing. These basic tasks are described in this section.

7.2.1 Identifying

Any service will ultimately be a composition of parts. These parts consist of things like data sets, programs, parm or configuration files, transactions, definitions, and so on. A thorough inventory and review is necessary to identify all of the components required for the service instance. In this context, even the actions performed during the creation of a service can be considered components and treated as such. Any inputs for service definitions must also be identified. The goal is to clearly determine each step, resource, and dependency that is required for the creation of the service.

Figure 7-2 provides a sample service group, UC01. This service group must have files created, and TCLASS, TRANSACTION, and URIMAP must be defined and installed. This process highlights several of the resources and steps that are necessary. However, certain inputs might be required from the consumer to define the service instance.

```
EX G(UC01)
ENTER COMMANDS
 NAME      TYPE         GROUP
 UC01VSAM  FILE         UC01
 TCLUC01   TRANCLASS    UC01
 UC01      TRANSACTION  UC01
 UC01      URIMAP       UC01
```

Figure 7-2 UC01 service group

These inputs, among others, might be required from the consumer to define the service instance:

- Tenant identification information
- Organizational information
- Availability requirements
- Volume projections

Other behind-the-scenes components might be required, such as the following:

- Parm updates
- Reporting adjustments
- Notifications to system administrators

7.2.2 Programming

After all components and steps have been identified, the creation of the service can be performed programmatically. The service can be created in various ways by using the tools, languages, and utilities that are most appropriate for the task at hand. This process can include a series of JCL steps, a REXX script, COBOL programs, or some combination of these and other options. These selections depend on individual use cases and requirements. Details such as the type of runtime environment being used, the particular actions to perform, and the type of resources to define determine the appropriate approach and tools to use.

Depiction of a complete automation process is not feasible in this book. Table 7-1 provides a summary of the types of programmatic resources and actions with examples that are part of the programming aspect of this description.

Table 7-1 Programmatic resources, actions, and examples

Resource/Action	**Examples**
Data sets, files	IEBGENER, IDCAMS, TSO ALLOC
System commands	REXX, EMCS
Parm updates	ISPF macros, IEBGENER, Execio
Control flow	JCL, REXX, COBOL
CICS resources	DFHCSDUP, RDO, CEDA, SPI

7.2.3 Exposing

Upon successful automation of the service creation, the capability needs to be made available to consumers. Generally, use standards-based web services as the channel for invocation of the service provisioning to ensure accessibility to the broadest range of clients. Many of the concepts covered in Chapter 5, “Ubiquitous access: ReST enabling your runtime” on page 41 are applicable to your efforts to expose the service.

A request for service provisioning will likely require that various pieces of information be passed to the system to enable proper configuration of the service instance. Information such as tenant IDs, organizational information, or SLA requirements might need to be passed by the consumer into the automation service, so a standard data interchange mechanism like JSON should be supported.

A final piece of this cloud characteristic that might be relevant is the consumer entry point. Although some service requests can come programmatically, most requests will likely come directly from consumers interacting with some form of online menu or catalog of service offerings. A wide array of products exists that might satisfy this purpose.

The entry point can be a service catalog system, an orchestrator, or a process or workflow system as depicted in Figure 7-3. Service providers can develop their own front ends for provisioning requests. The details of development are beyond the scope of this book. This function is usually provided by third party solutions. However, adherence to web standards increases the likelihood of interoperability with other solutions.

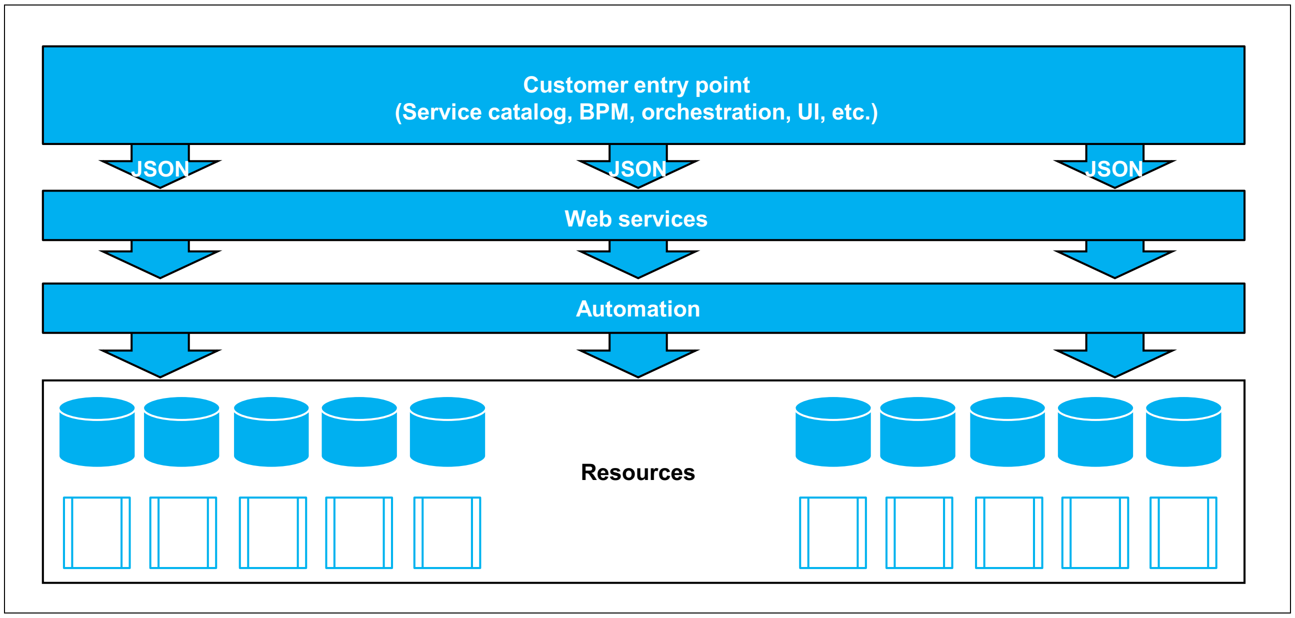


Figure 7-3 Overall view of provisioning

7.3 Experiences

Self service was the last characteristic addressed and implemented by Walmart. The first service deployed required resources to be defined semi-manually by and for early adopters.

From the onset, parts of the service creation were performed with small, limited function scripts or jobs. Over several iterations, the components and processes became more familiar, and each piece was automated and included in an overall control flow. The experience gained in this approach allowed for the inclusion of automated provisioning as part of the design for future services

The Walmart developers initially chose to develop their own portal from standard components and hosted entirely on z/OS to maintain scope of control on the provisioning entry point.

The portal was an HTML-based web app that served as a basic catalog into the underlying automated provisioning functions for their services. Using form-based web services, the portal also included validation features based on JavaScript in the forms to help enforce service options such as naming conventions.

Eventually, the provisioning services were modified to be JSON-enabled to promote extensibility. Walmart's z/OS cloud services can now be provisioned from other service catalogs or workflow solutions, such as IBM Business Process Manager (BPM), using JSON requests in addition to continued manual provisioning through the portal

Walmart developers determined that any number of components can be used to provide back-end provisioning functions. This provisioning process could be as simple and basic as submitting JCL to run a job consisting of the necessary steps to provision resources. The JCL submission approach effectively results in asynchronous delivery of resources. Although not automatic, the JCL submission approach is more valuable and efficient than a series of manual actions.

Developers at Walmart elected to approach service delivery through synchronous means. They used CICS as the runtime environment and used a combination of COBOL and assembly language programs to perform provisioning activities. The process returned the ReSTful endpoints (which also facilitated Development and QA environments) immediately to the consumer on completion of the HTTP request. At the time of writing, Walmart has also automated provisioning in production environments. The automated provisioning still requires human intervention from operations to complete the request.

7.4 Evolving your self-service strategy

If you wait until you have everything figured out it will be too late. You will have missed your opportunity to become essential to the business. No one expects you to have this all figured out on day one of your journey. Determine the minimum viable product and improve from there, rather than waiting for the mythical perfect solution.

Successful service providers build their service strategy around the actual wants and needs of the consumer. After your cloud strategy takes shape, the real demands will reveal themselves.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- *Deploying a Cloud on IBM System z*, REDP-4711
- *How Walmart Became a Cloud Services Provider with IBM CICS*, SG24-8347

You can search for, view, download or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- The NIST Definition of Cloud Computing

  http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf
- Competitive advantage is in the clouds: A discussion with Frank De Gilio

  http://ibmcai.com/2015/08/04/competitive-advantage-is-in-the-clouds-a-discussion-with-frank-degilio/
- IBM z/OS documentation

  http://www-01.ibm.com/support/knowledgecenter/SSLTBW_2.2.0/com.ibm.zos.v2r2/en/homepage.html

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

SG24-8324-00

ISBN 0738441538

Printed in U.S.A.